Huffman Codes

Encoding messages

- Encode a message composed of a string of characters
- Codes used by computer systems
  - ASCII
    - uses 8 bits per character
    - can encode 256 characters
  - Unicode
    - 16 bits per character
    - can encode 65536 characters
    - includes all characters encoded by ASCII
- ASCII and Unicode are *fixed-length codes*
  - all characters represented by same number of bits

Problems

- Suppose that we want to encode a message constructed from the symbols **A**, **B**, **C**, **D**, and **E** using a fixed-length code
  - How many bits are required to encode each symbol?
    - at least 3 bits are required as 5 characters are there(**A**, **B**, **C**, **D**, and **E**)
    - 2 bits are not enough (can only encode four symbols)
  - How many bits are required to encode the message **DEAACAAAAABA**?
    - there are twelve symbols, each requires 3 bits
    - 12*3 = 36 bits are required

Drawbacks of fixed-length codes

- Wasted space
    - inefficient for plain-text messages containing only ASCII characters
- Same number of bits used to represent all characters
    - ‘a’ and ‘e’ occur more frequently than ‘q’ and ‘z’

- **Potential solution**: use variable-length codes
    - variable number of bits to represent characters when frequency of occurrence is known
    - short codes for characters that occur frequently

Advantages of variable-length codes

- The advantage of variable-length codes over fixed-length is short codes can be given to characters that occur frequently
  - on average, the length of the encoded message is less than fixed-length encoding
- **Potential problem:** how do we know where one character ends and another begins?
  - not a problem if number of bits is fixed!

A = 00
B = 01
C = 10
D = 11

0010110111001111111111

A C D B A D D D D D

- Example:

Symbol	Code
P	000
Q	11
R	01
S	001
T	10

01001101100010

R S T Q P T

Symbol	**Code**
P	0
Q	1
R	01
S	10
T	11

- The pattern **1110** can be decoded as **QQQP**, **QTP**, **QQS**, or **TS**

What code to use?

- Question: Is there a variable-length code that makes the most efficient use of space?

Answer: Yes!

Huffman coding tree

- Binary tree
  - each leaf contains symbol (character)
  - label edge from node to left child with 0
  - label edge from node to right child with 1
- Code for any symbol obtained by following path from root to the leaf containing symbol
- Code has prefix property
  - leaf node cannot appear on path to another leaf
  - *note*: fixed-length codes are represented by a complete Huffman tree and clearly have the prefix property

Building a Huffman tree

- Find frequencies of each symbol occurring in message
- Begin with a forest of single node trees
  - each contain symbol and its frequency
- Do recursively
  - select two trees with smallest frequency at the root
  - produce a new binary tree with the selected trees as children and store the sum of their frequencies in the root
- Recursion ends when there is one tree
  - this is the Huffman coding tree

Example

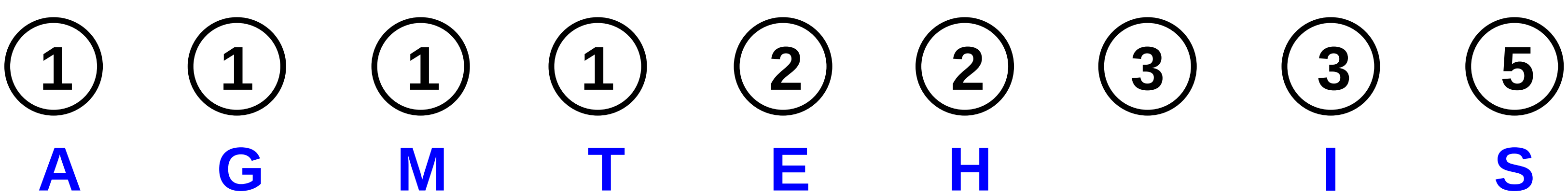
- Build the Huffman coding tree for the message

This is his message

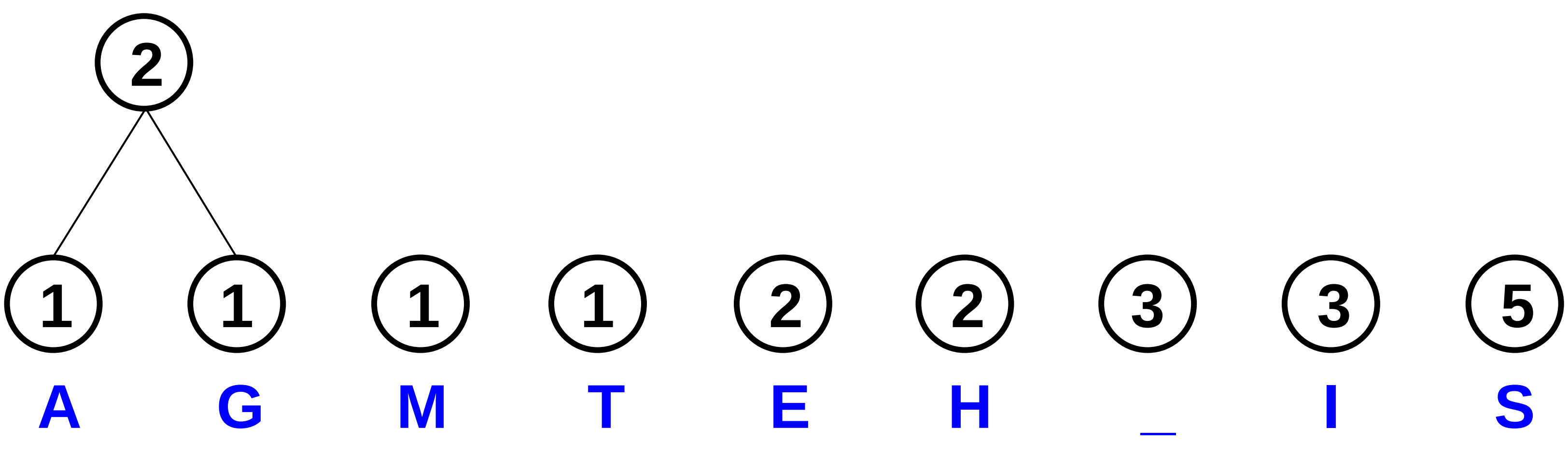
- Character frequencies

A	G	M	T	E	H	_	I	S
1	1	1	1	2	2	3	3	5

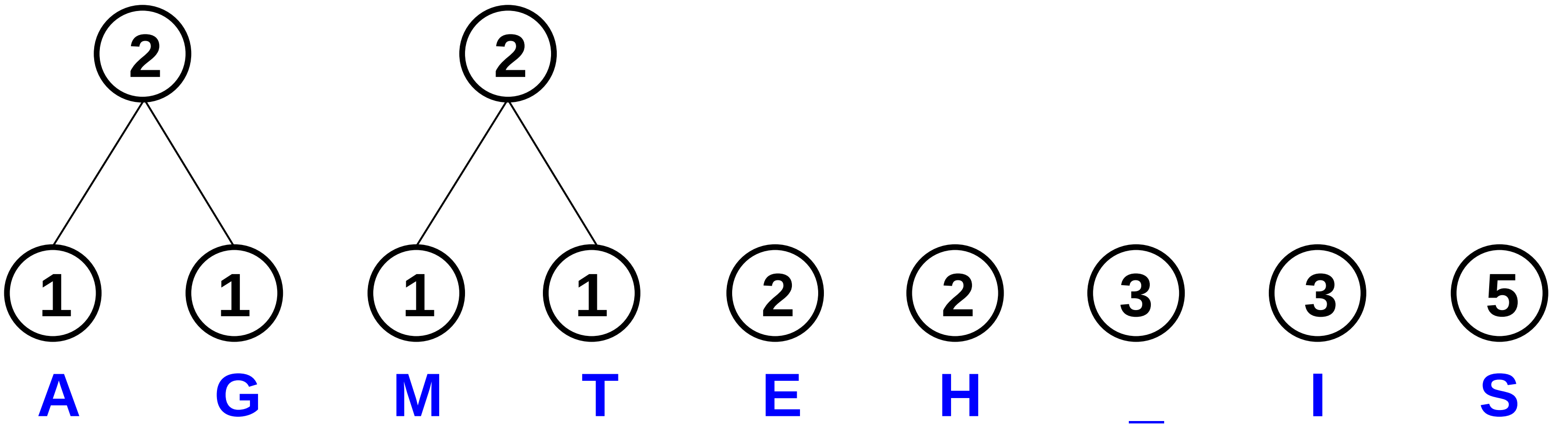
- Begin with forest of single trees



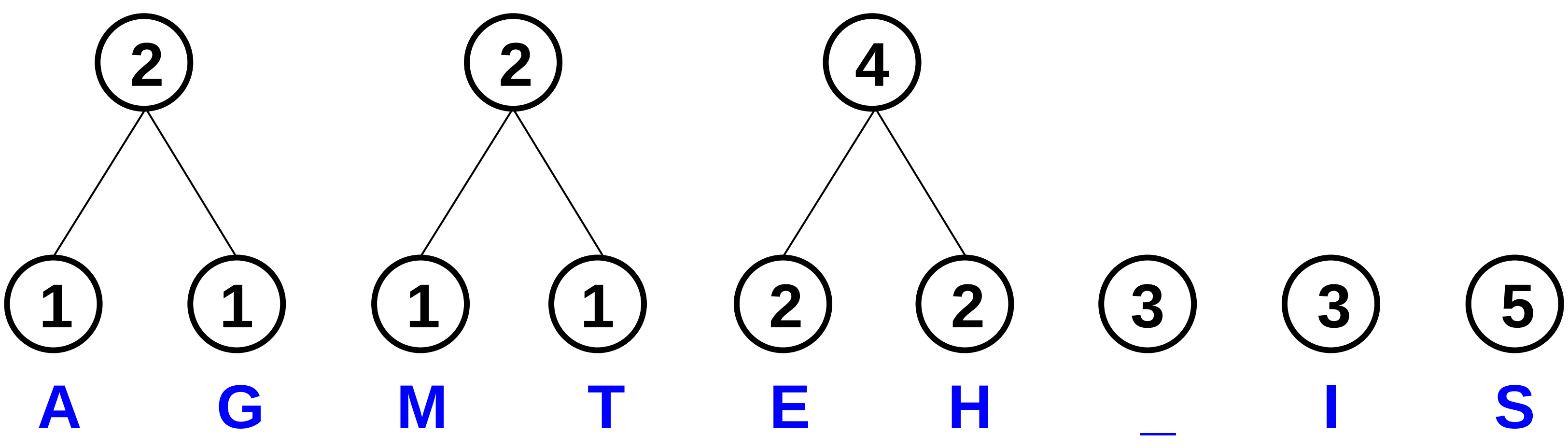
Step 1



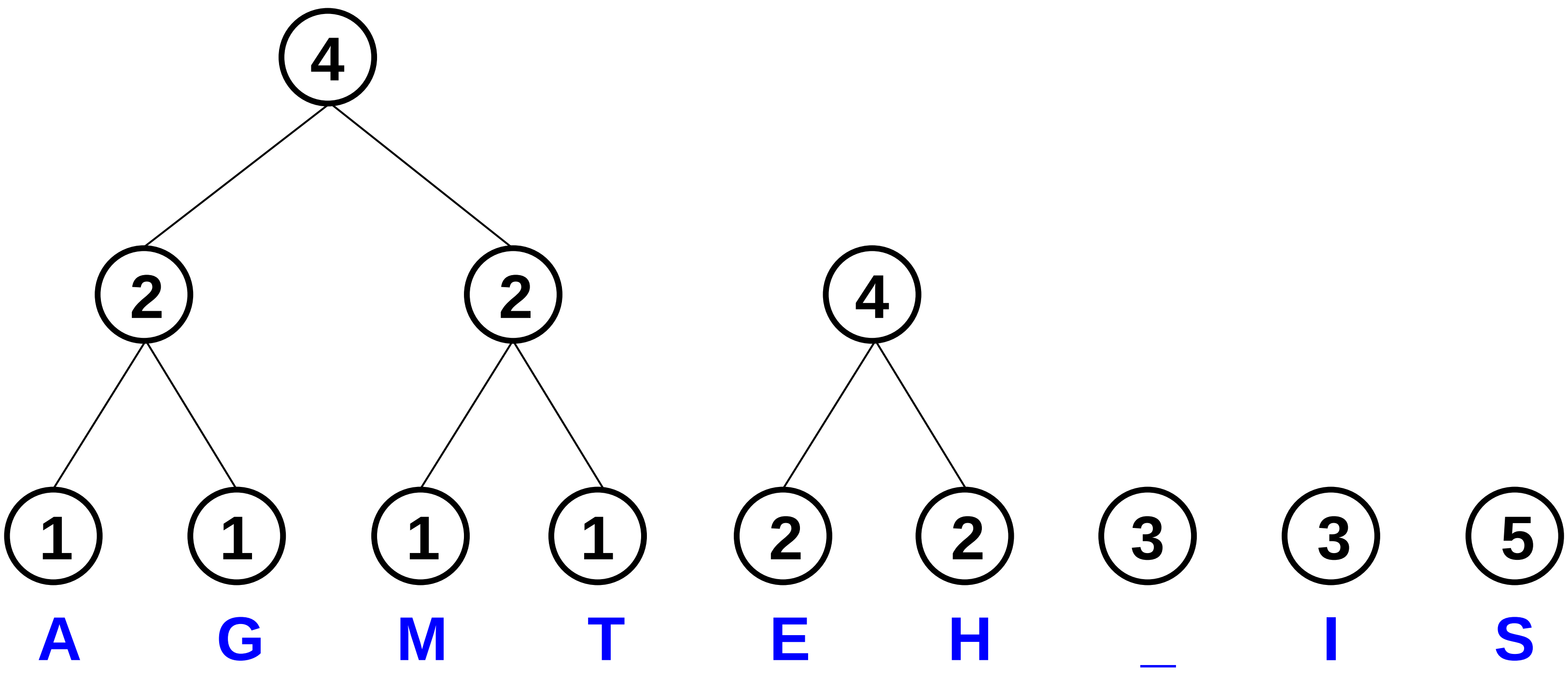
Step 2



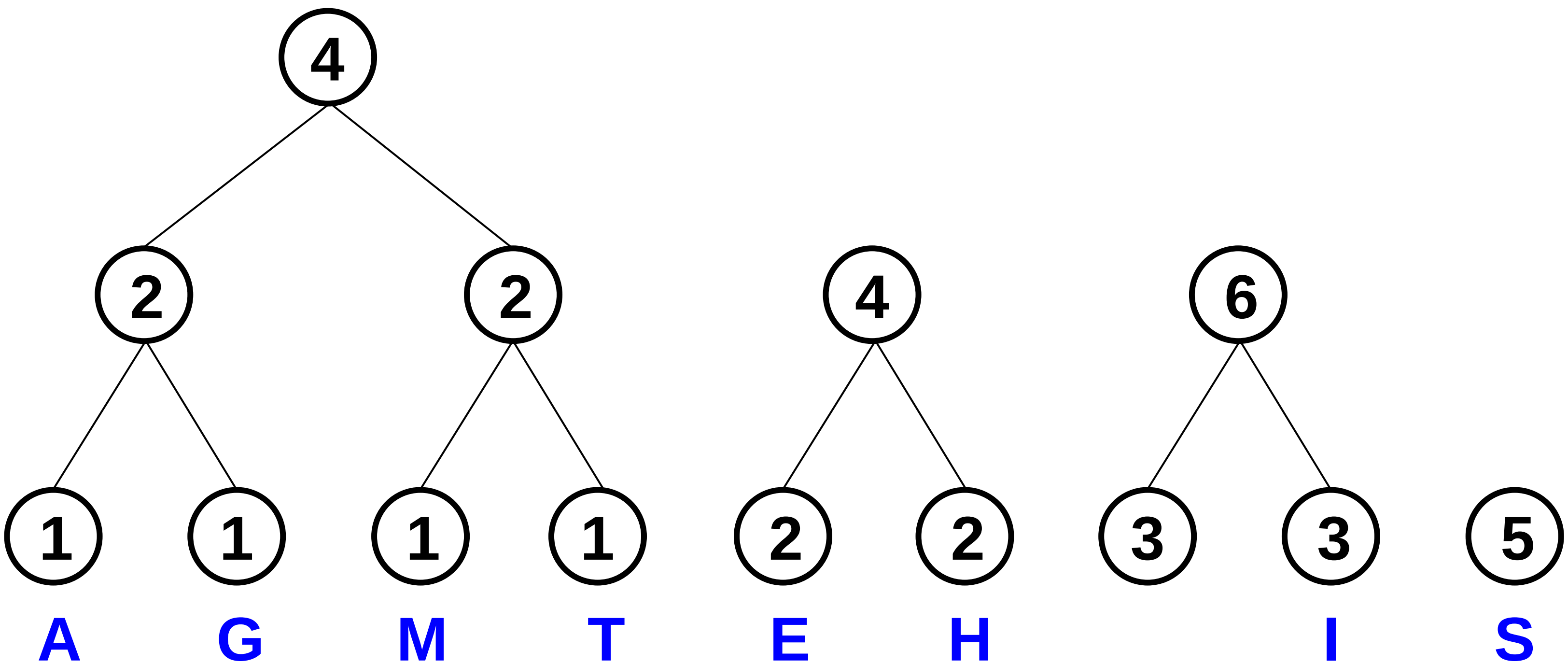
Step 3



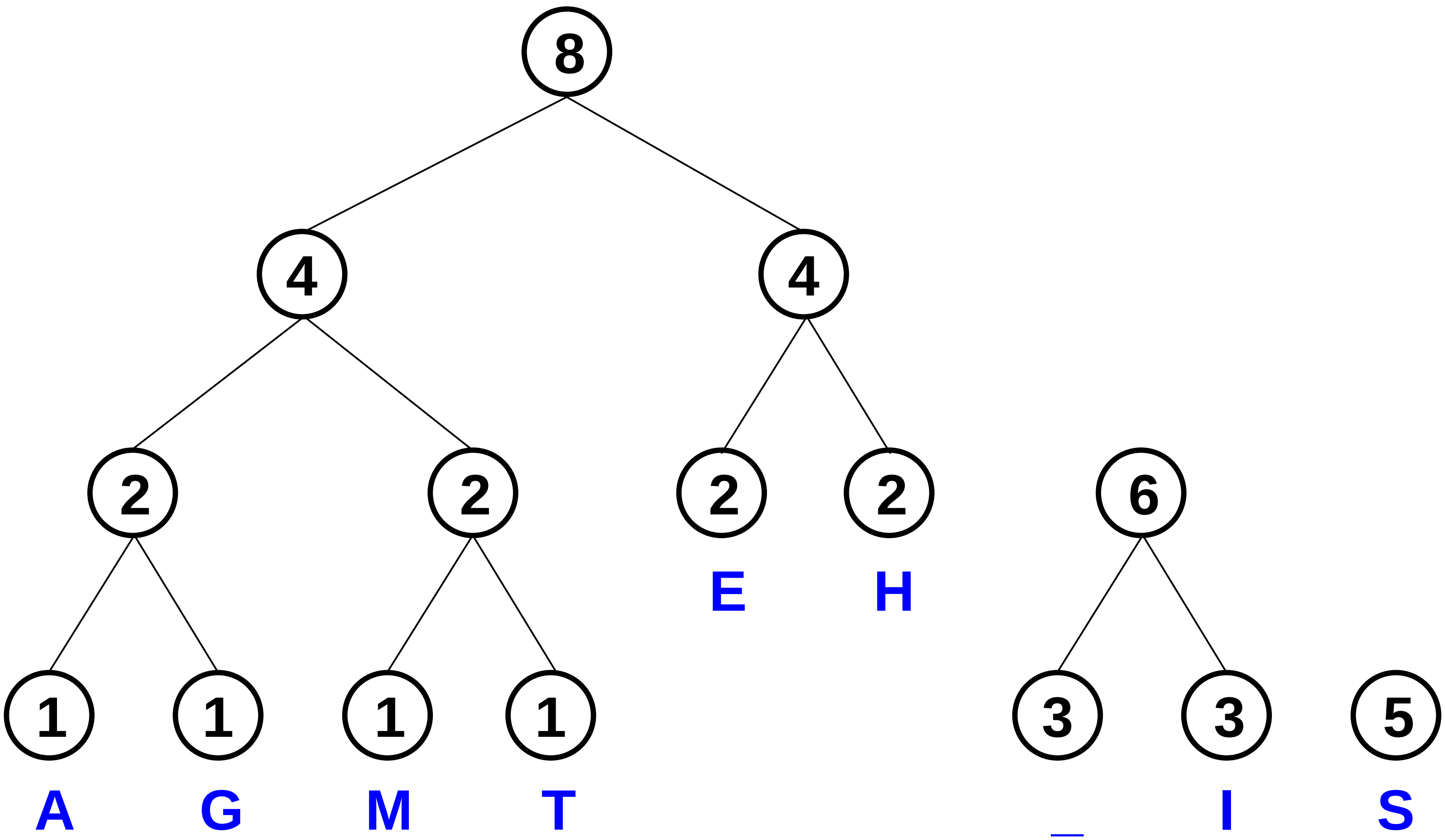
Step 4



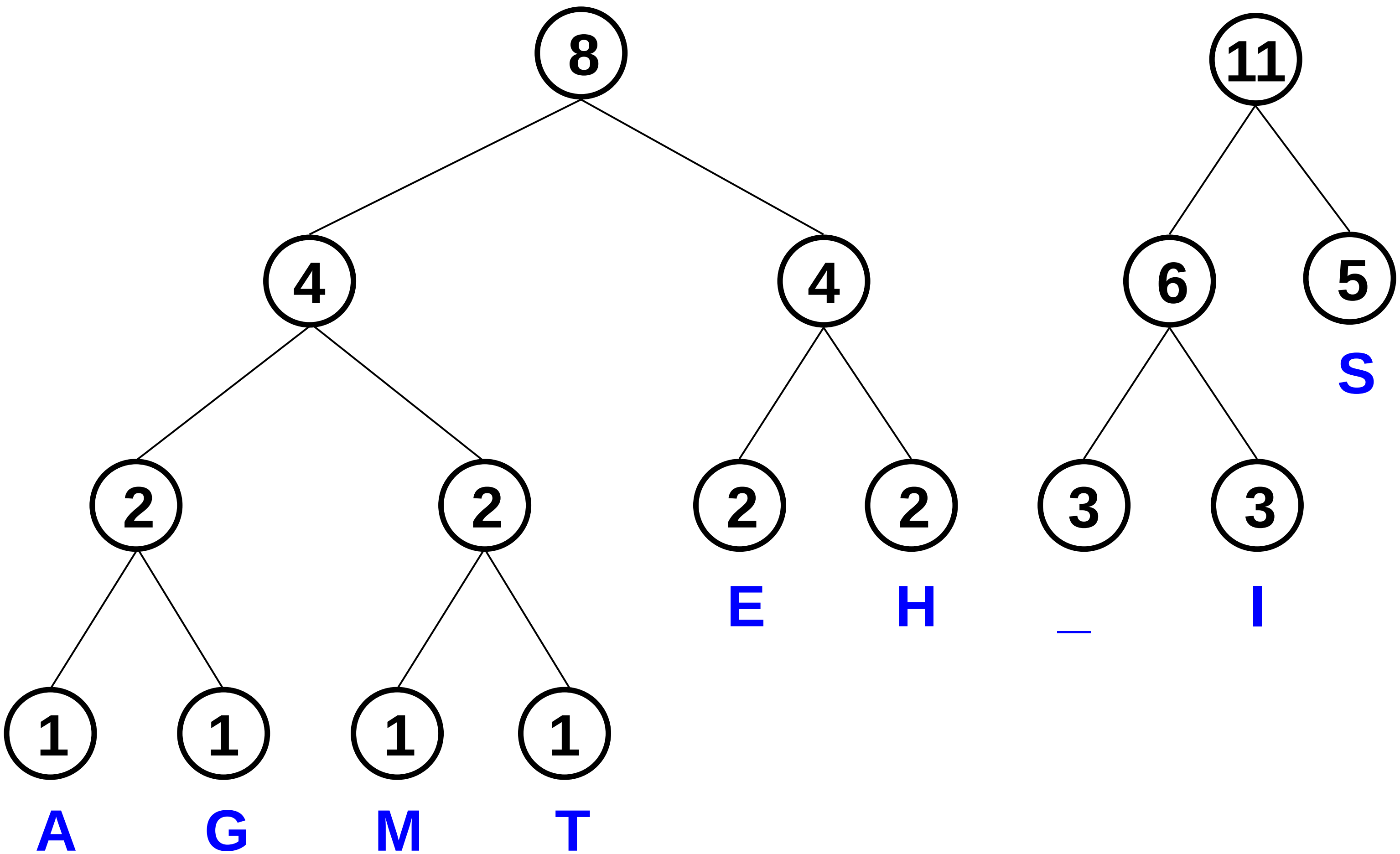
Step 5



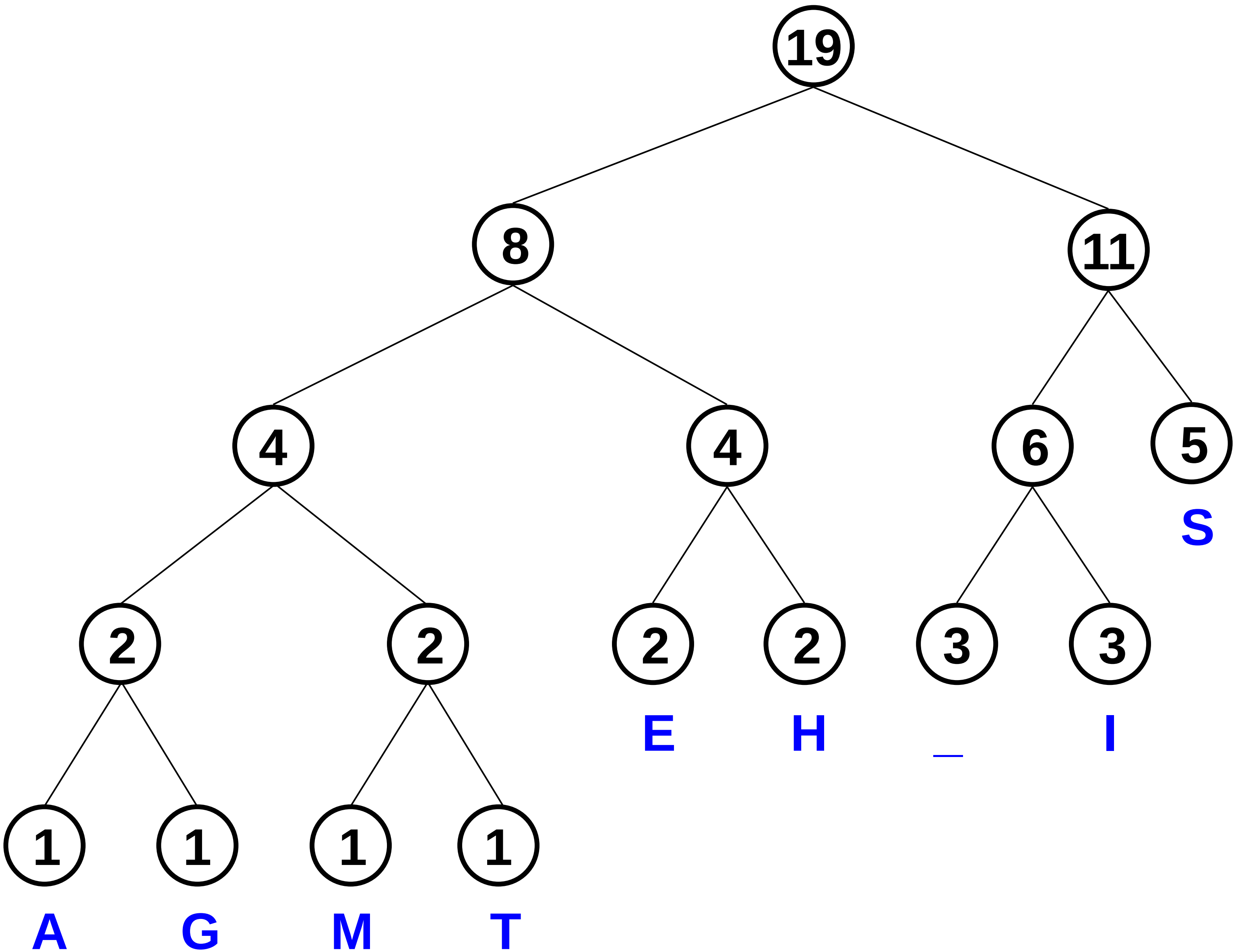
Step 6



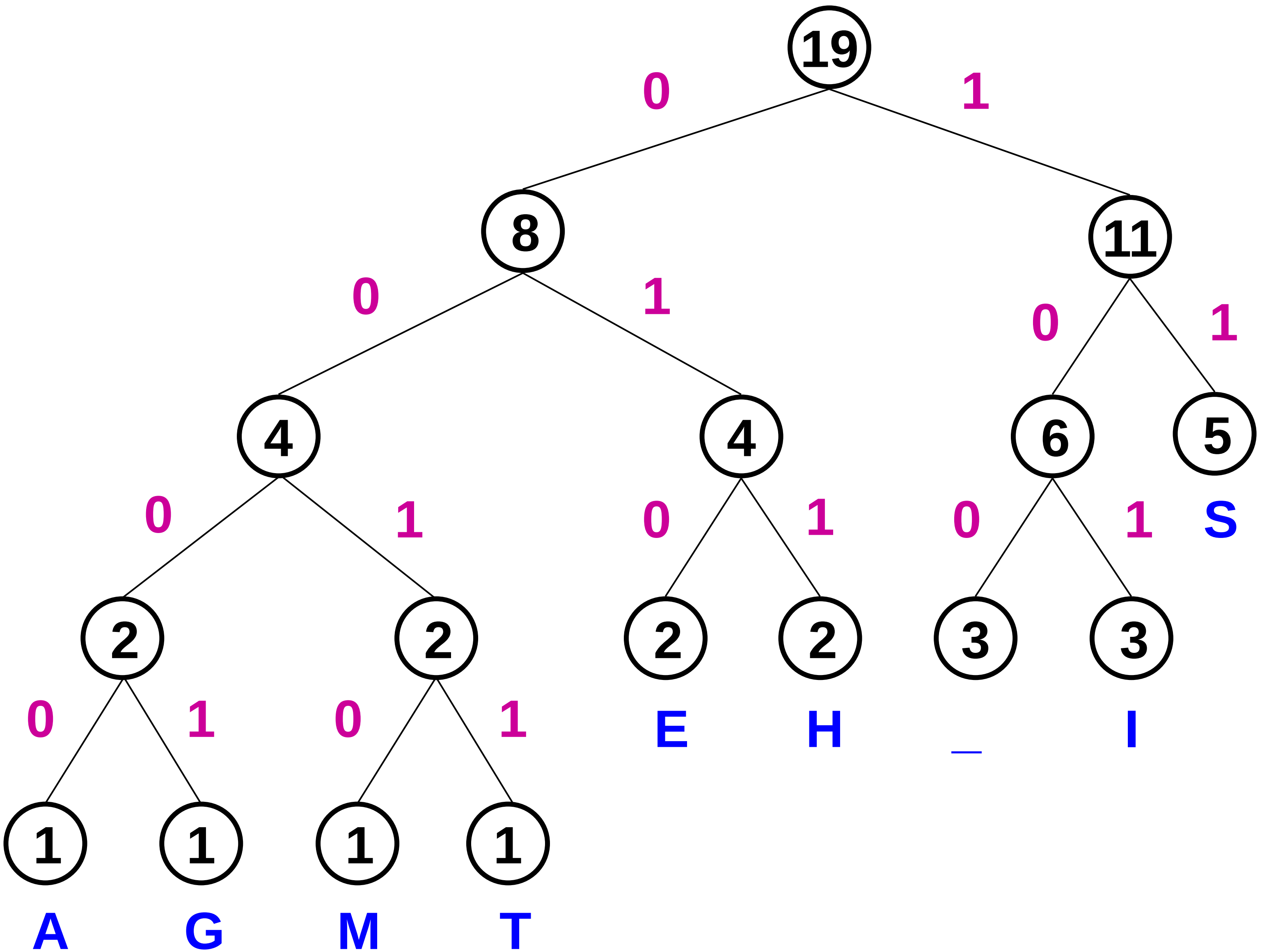
Step 7



Step 8



Label edges



Huffman code & encoded message

This is his message

S	11
E	010
H	011
_	100
I	101
A	0000
G	0001
M	0010
T	0011

00110111011110010111100011101111000010010111100000001010

Practice problem

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

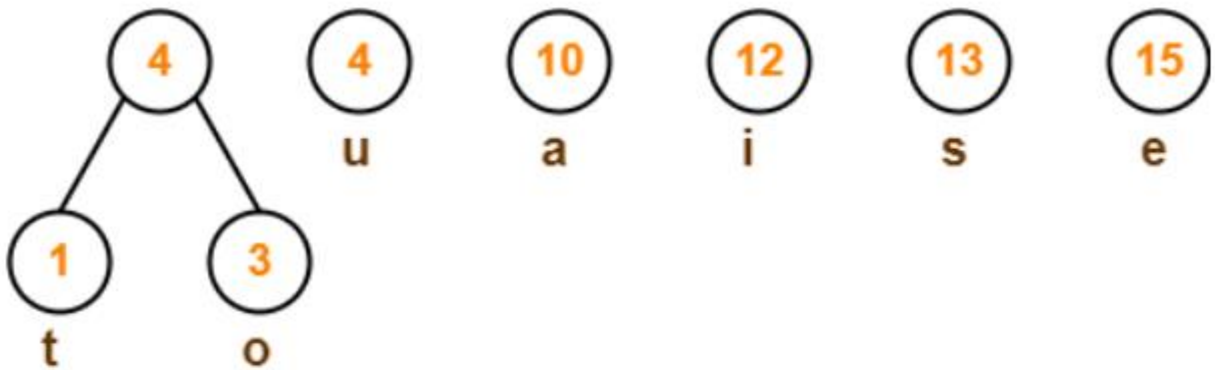
1. Huffman Code for each character
2. Average code length
3. Length of Huffman encoded message (in bits)

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

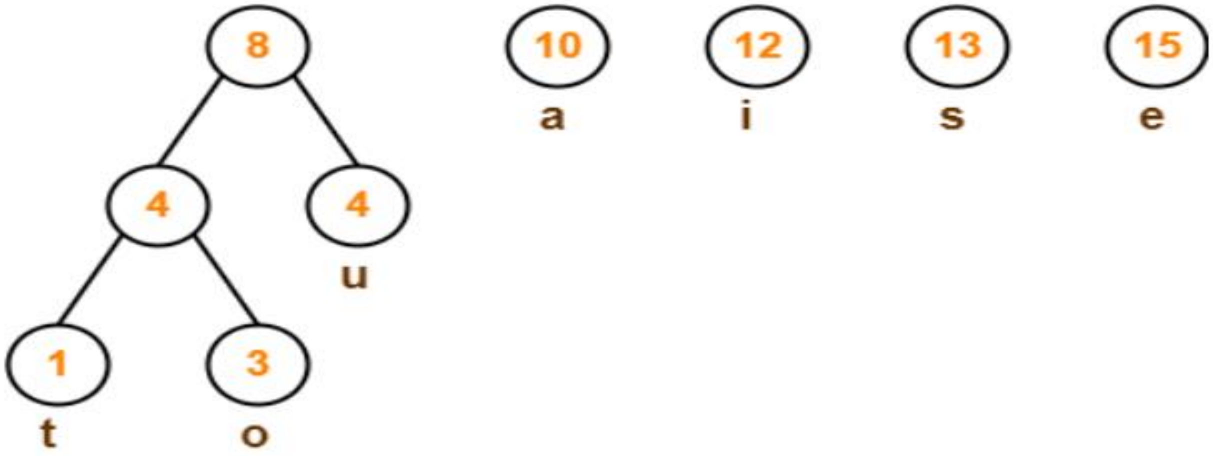
Huffman Tree is constructed in the following steps-

Step 1:

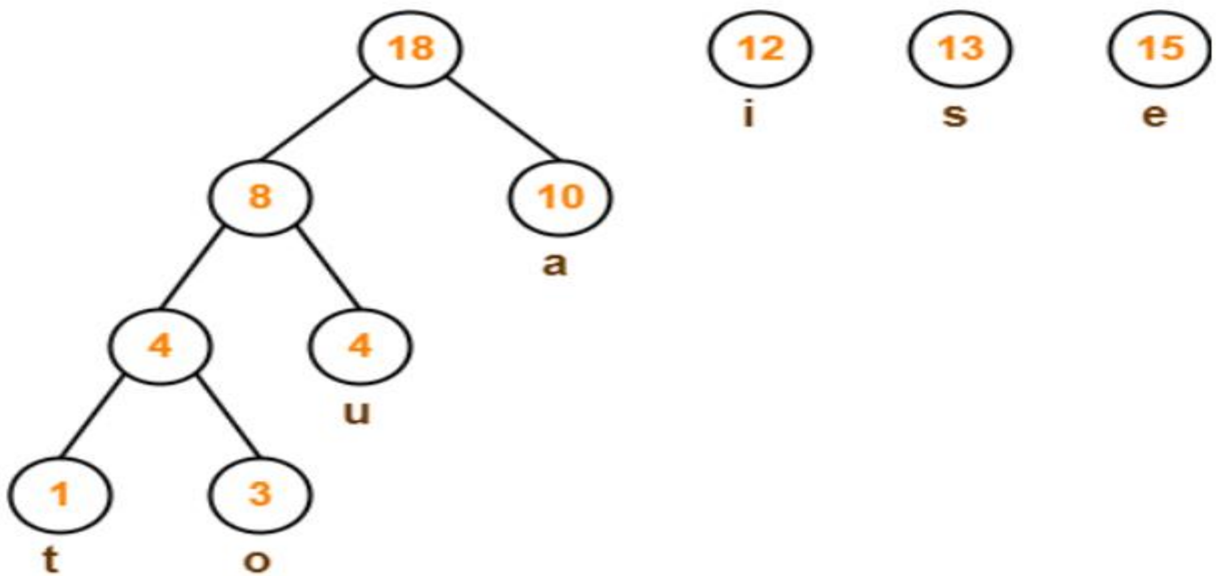
Step-02:



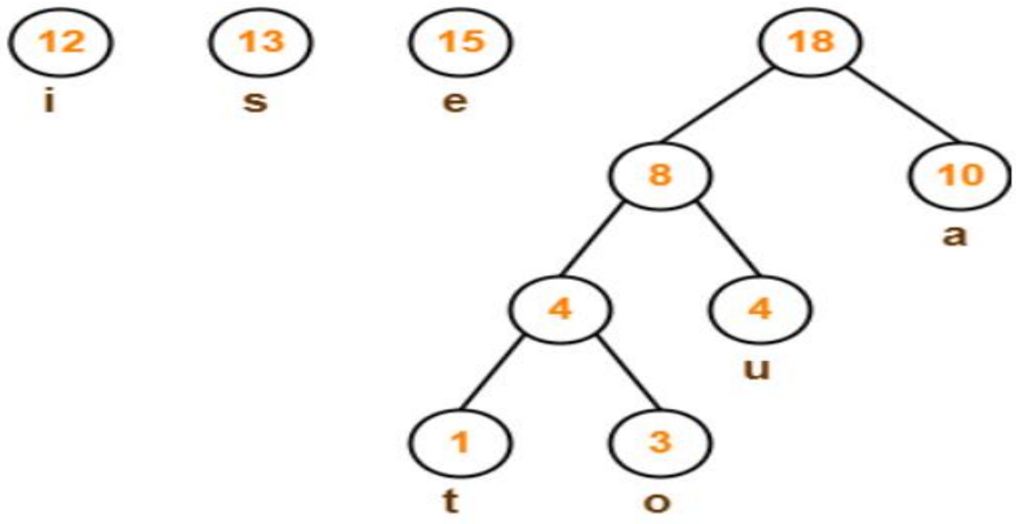
Step-03:



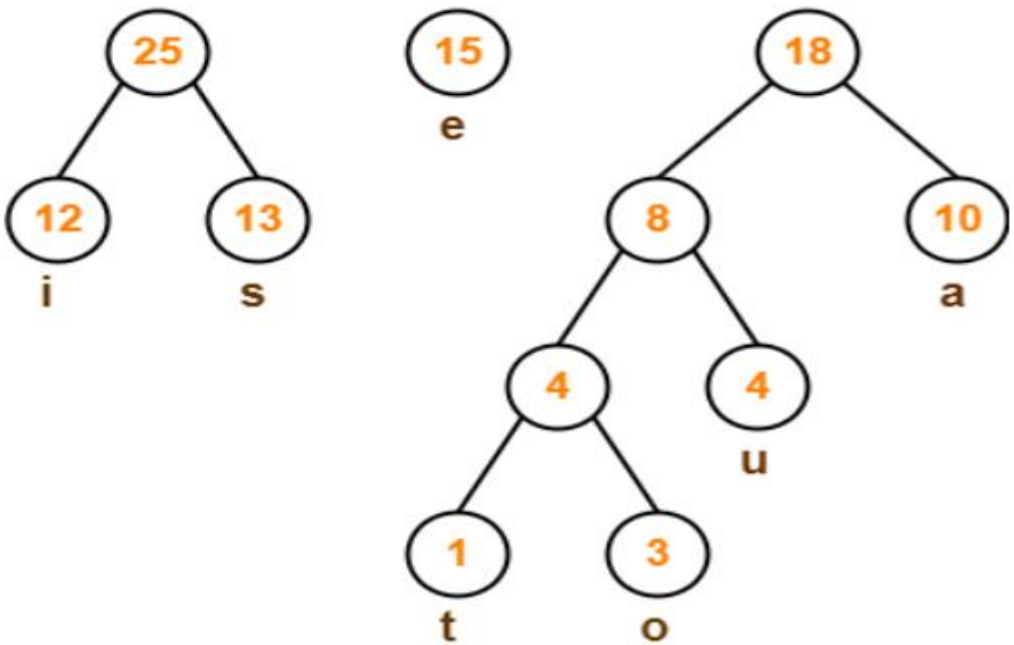
Step-04:



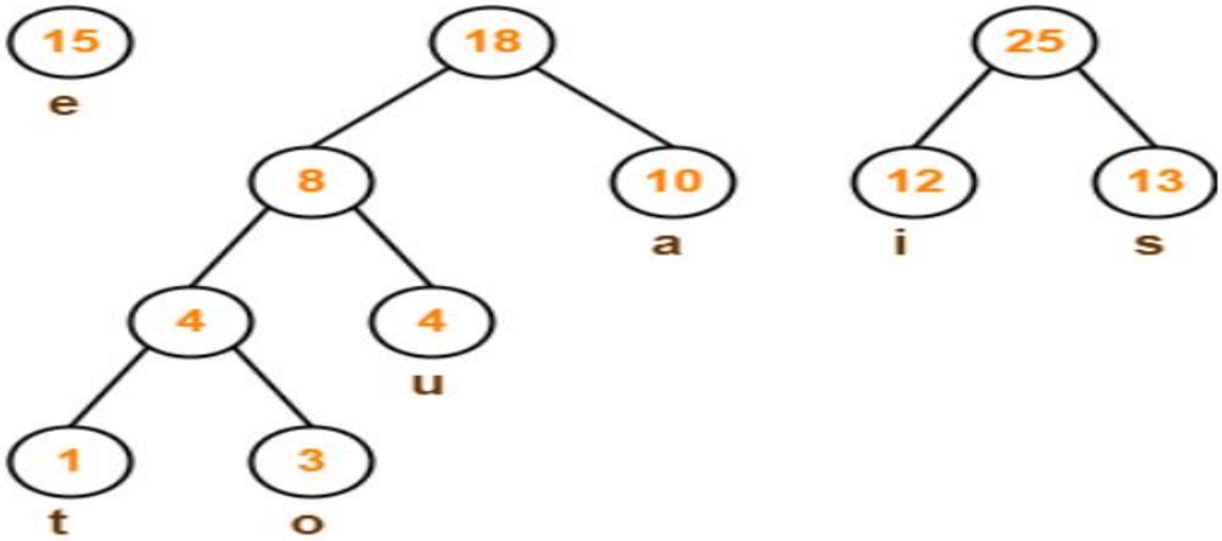
Step-05:



Step-6 :

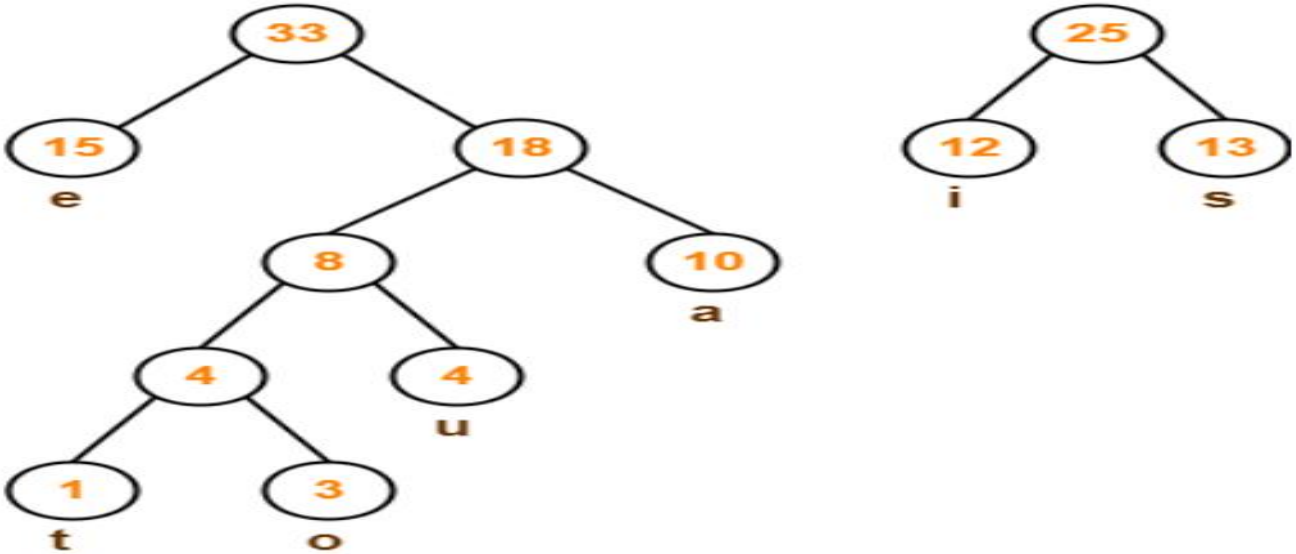


Step-7:



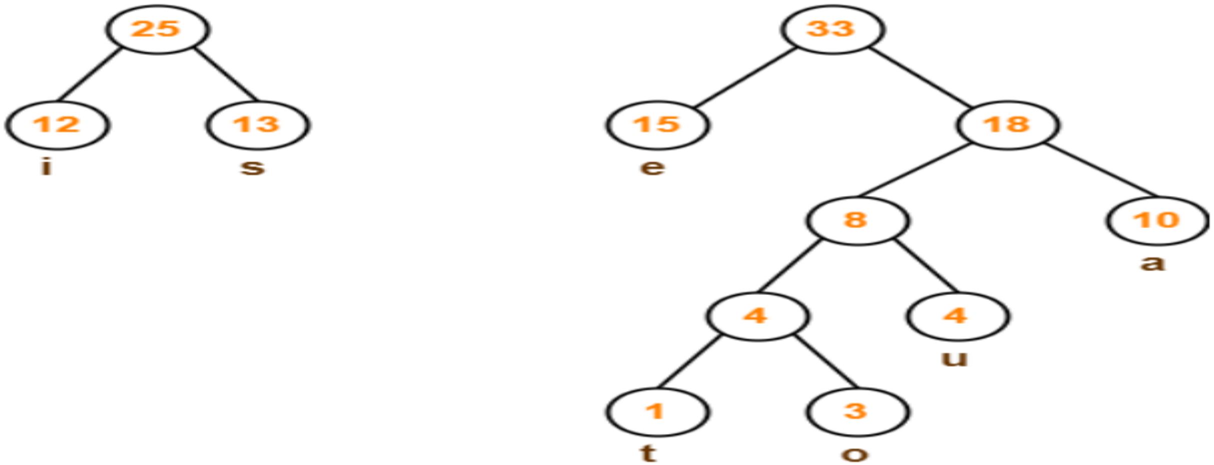
15
e
18
8
10
a
4
4
u
1
t
3
o
25
12
i
13
s

Step-8:

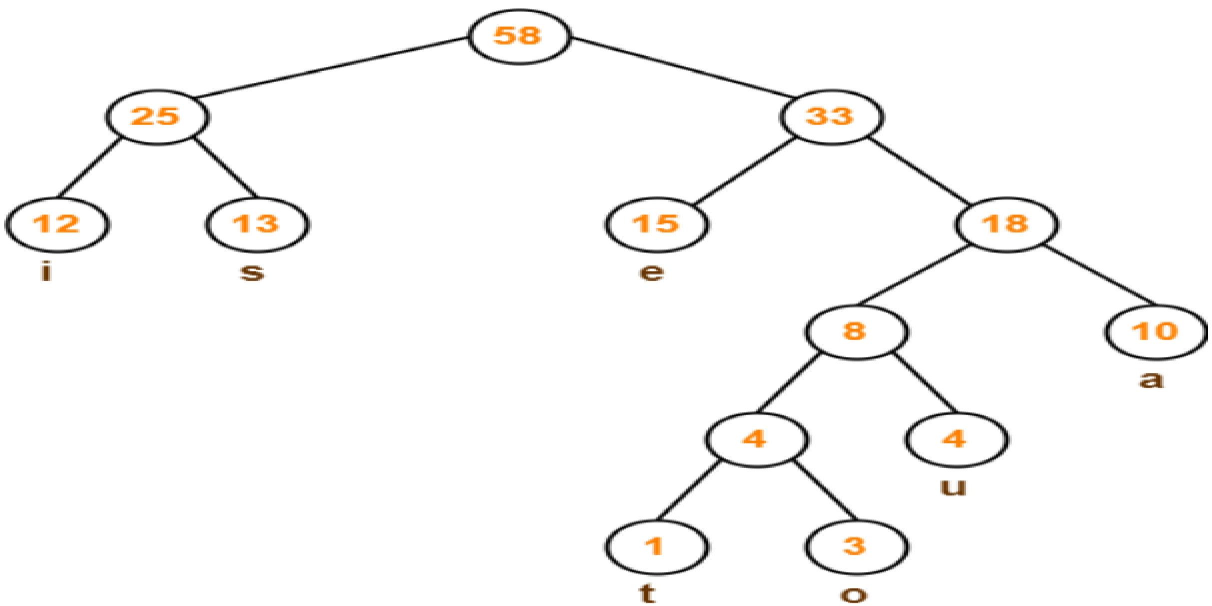


33
15
e
18
8
10
a
4
4
u
1
t
3
o
25
12
i
13
s

Step-8: rearranged in ascending order



Step-9: Final Huffman Tree



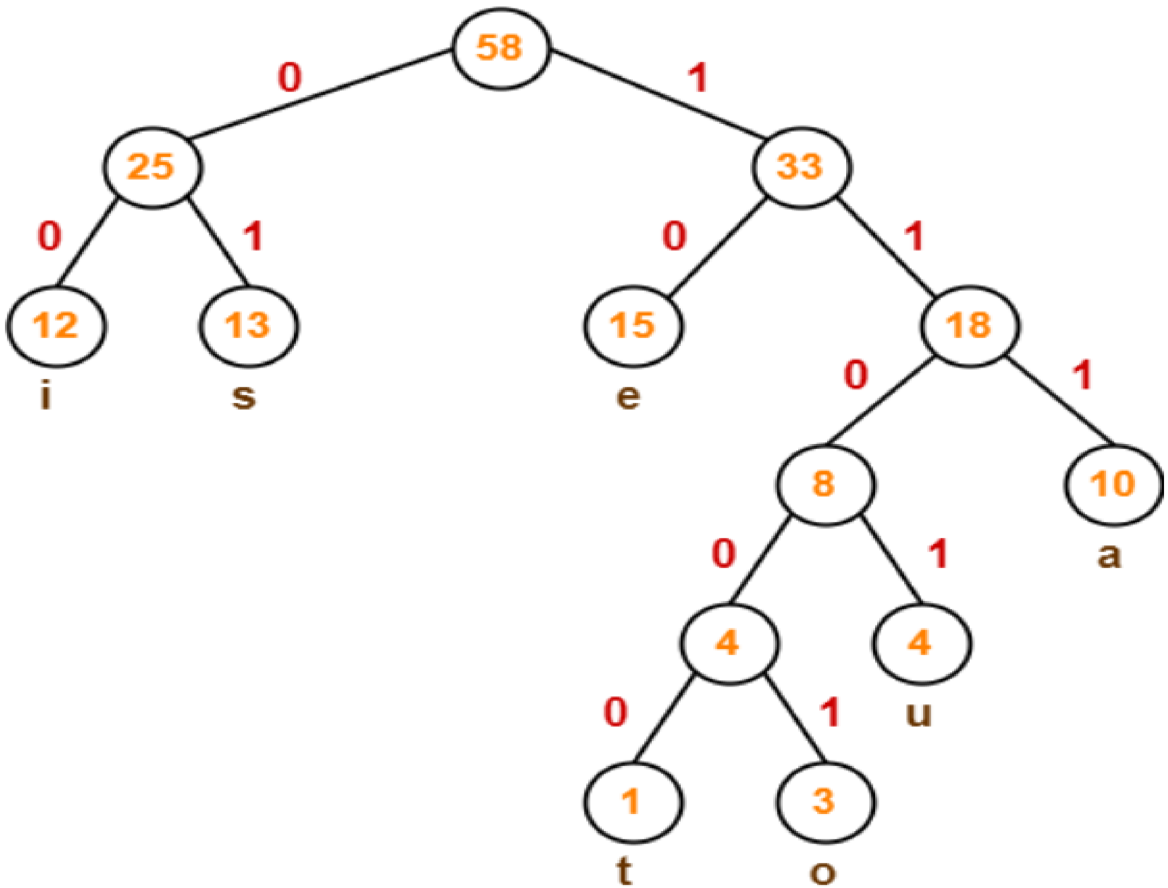
Huffman Tree

Now,

•We assign weight to all the edges of the constructed Huffman Tree.

•Let us assign weight ‘0’ to the left edges and weight ‘1’ to the right edges.

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



Huffman Tree

1. Huffman Code For Characters-

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

From here, we can observe:

- Characters occurring less frequently in the text are assigned the larger code.
- Characters occurring more frequently in the text are assigned the smaller code.

2. Average Code Length-

$$\text{Average code length} = \frac{\sum (\text{frequency}_i \times \text{code length}_i)}{\sum (\text{frequency}_i)}$$

$$= \frac{\{(10 \times 3) + (15 \times 2) + (12 \times 2) + (3 \times 5) + (4 \times 4) + (13 \times 2) + (1 \times 5)\}}{(10 + 15 + 12 + 3 + 4 + 13 + 1)}$$

= 2.52

3. Length of Huffman Encoded Message-

Total number of bits in Huffman encoded message

= Total number of characters in the message x Average code length per character

= 58 x 2.52

= 146.16

≅ 147 bits