> "Today is cruel. Tomorrow is crueller. And the day after tomorrow is beautiful."
>
> \- Jack Ma

Lecture 25

Greedy Algorithms: Huffman Encoding & its Time Complexity, Activity Scheduling

Huffman Encoding: Definition

- Huffman Coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.
- The most frequent character gets the smallest code and the least frequent character gets the largest code.

Huffman Encoding

- 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**.

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

Huffman Encoding: Important Steps

- The steps involved in the construction of Huffman Tree are as follows-
- Step-01:
  - Create a leaf node for each character of the text.-
  - Leaf node of a character contains the occurring frequency of that character.
- Step-02:
  - Arrange all the nodes in increasing order of their frequency value.
- Step-03:
  - Considering the first two nodes having minimum frequency,
  - Create a new internal node.
  - The frequency of this new node is the sum of frequency of those two nodes.
  - Make the first node as a left child and the other node as a right child of the newly created node.
- Step-04:
  - Keep repeating Step-02 and Step-03 until all the nodes form a single tree.
  - The tree finally obtained is the desired Huffman Tree.

Huffman Encoding: Important Formulas

(1)

$$\text{Average code length per character} = \frac{\Sigma\,(\text{frequency}_i \times \text{code length}_i)}{\Sigma\,\text{frequency}_i}$$

$$= \Sigma\,(\text{probability}_i \times \text{code length}_i)$$

(2)

Total number of bits in Huffman encoded message

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

$= \sum\,(\text{frequency}_i \times \text{Code length}_i)$

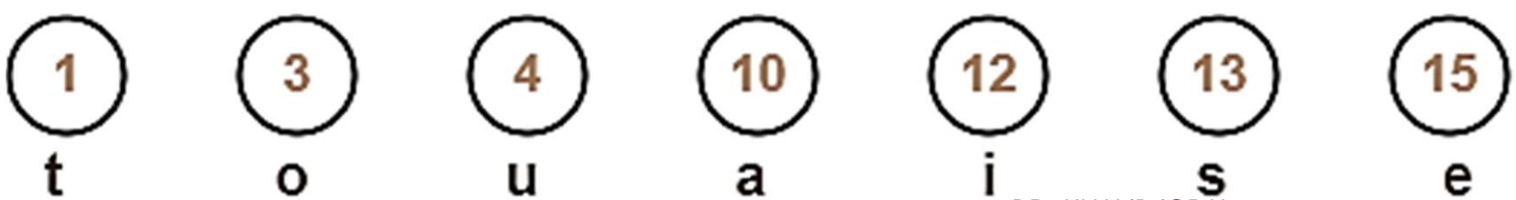
Huffman Encoding: Problem

- A file contains the following characters with the frequencies as shown. If Huffman Coding is used for data compression, determine-
  - Huffman Code for each character
  - Average code length
  - Length of Huffman encoded message (in bits)

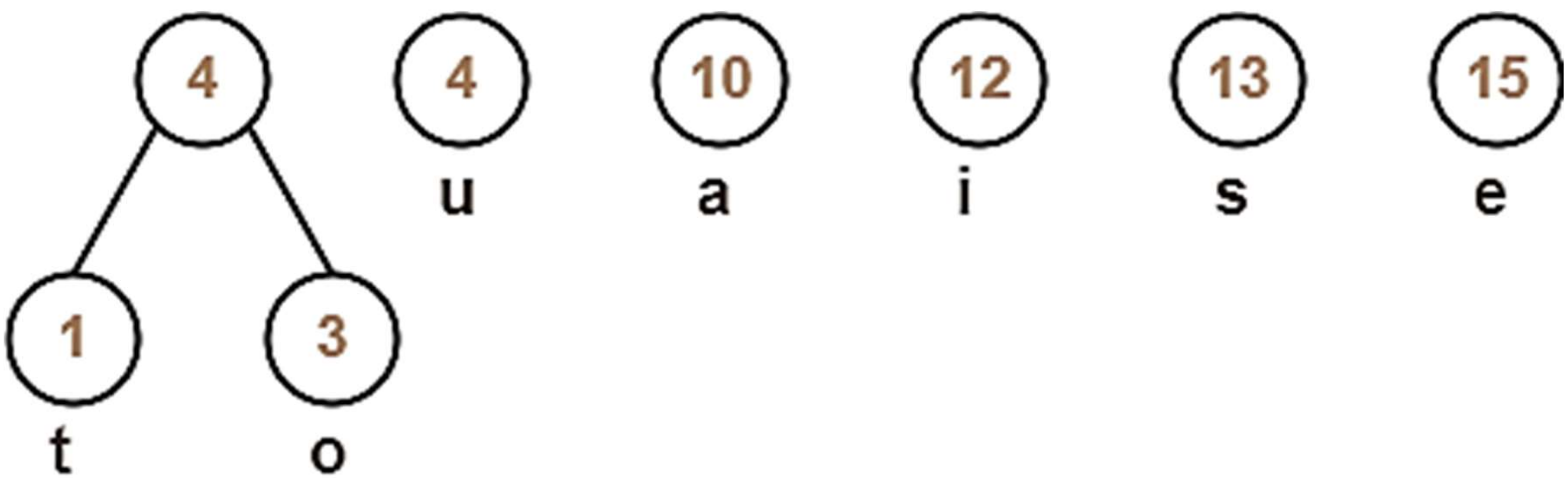
Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1

Solution

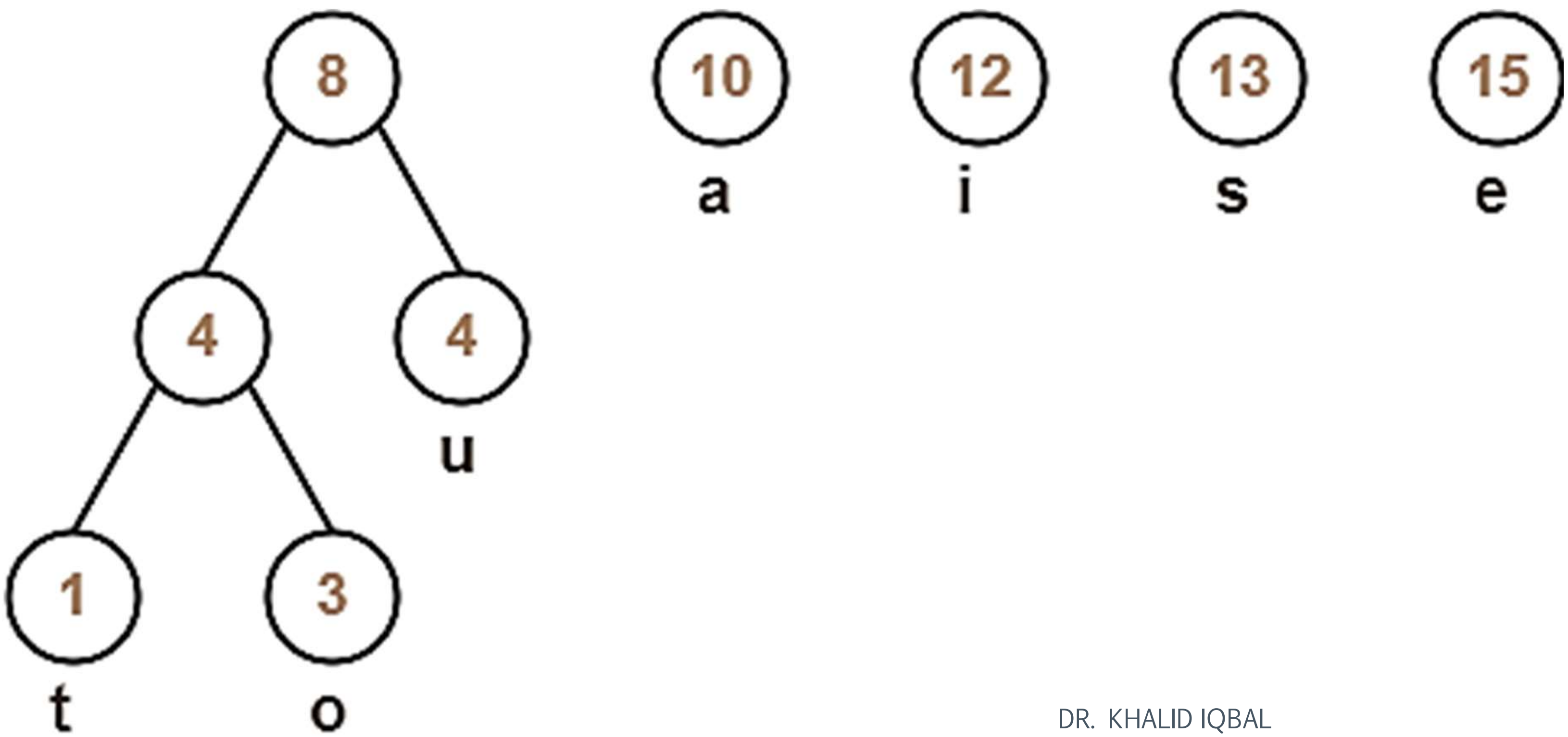
› Construct the Huffman Tree



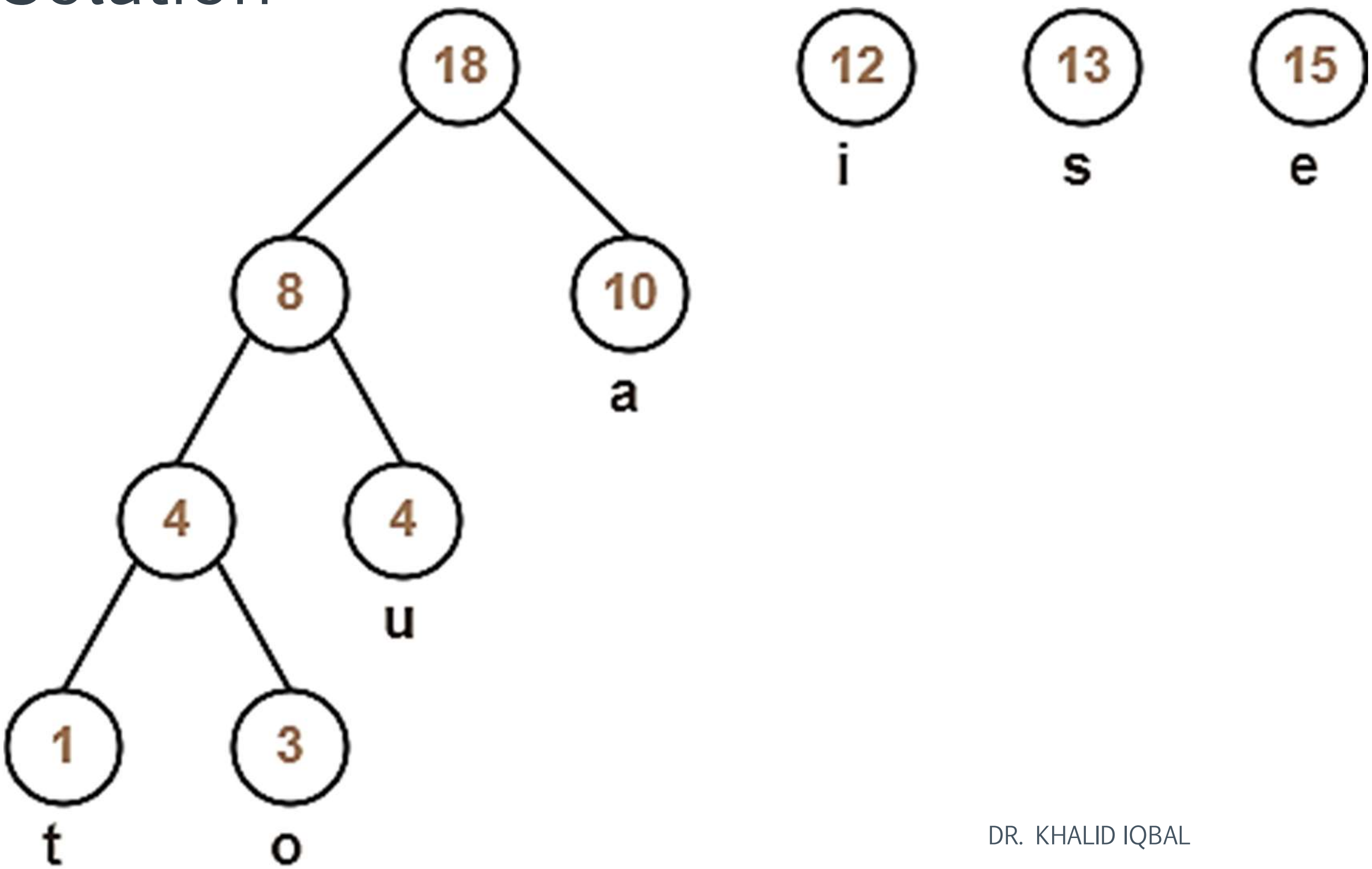
Solution



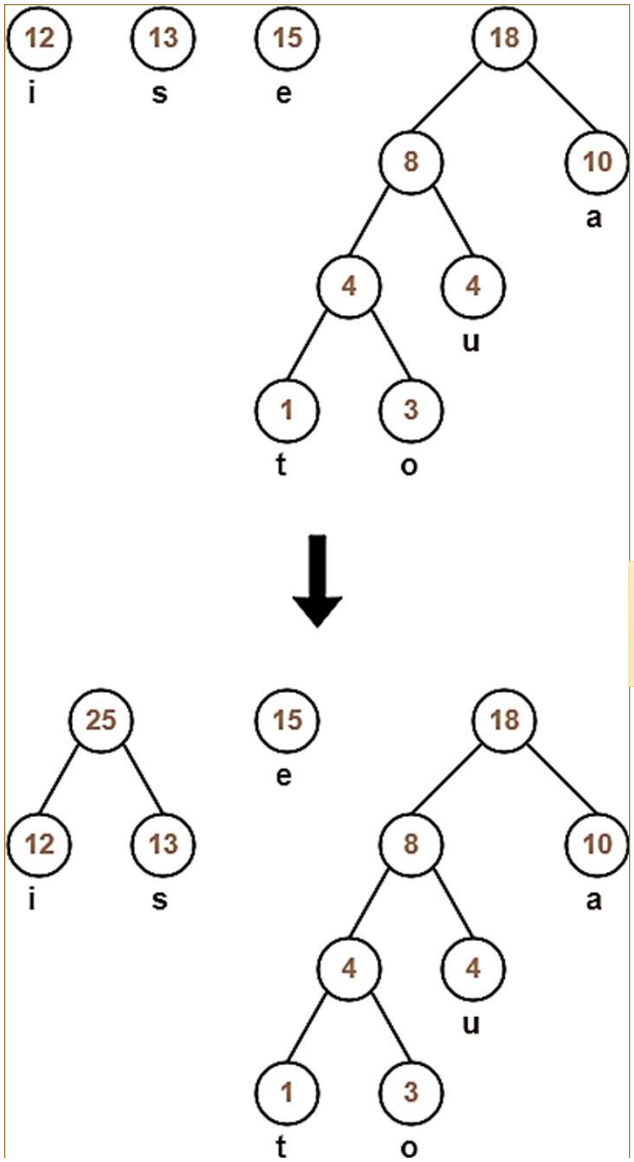
Solution

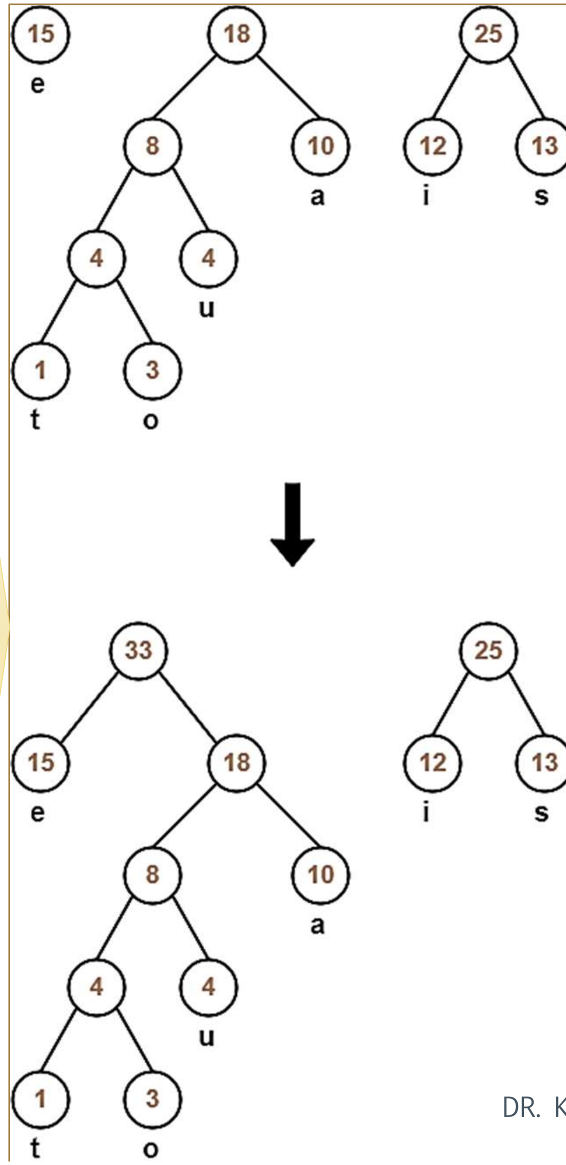


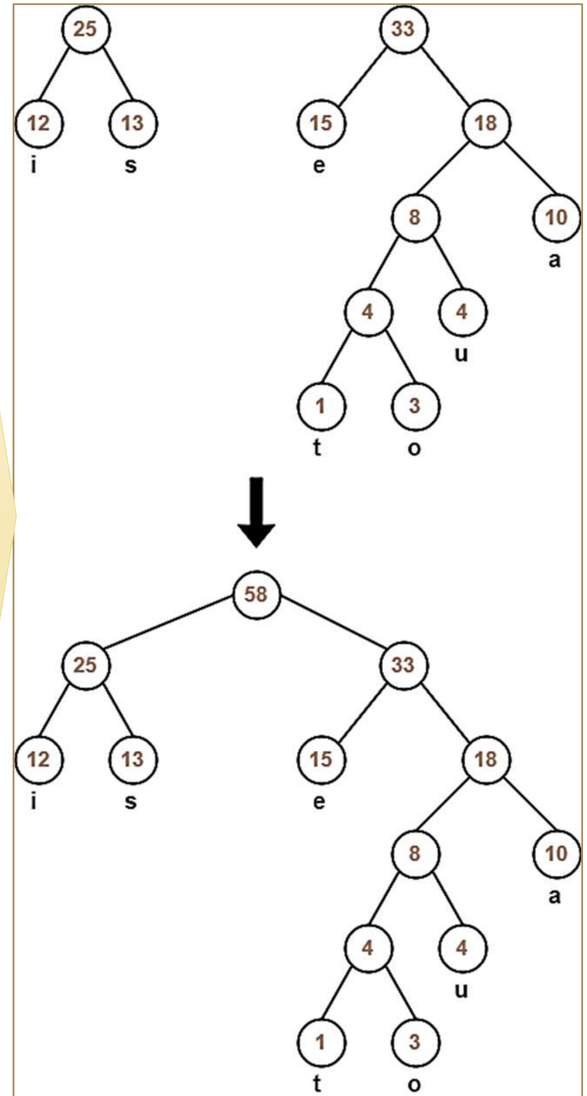
Solution



Solution







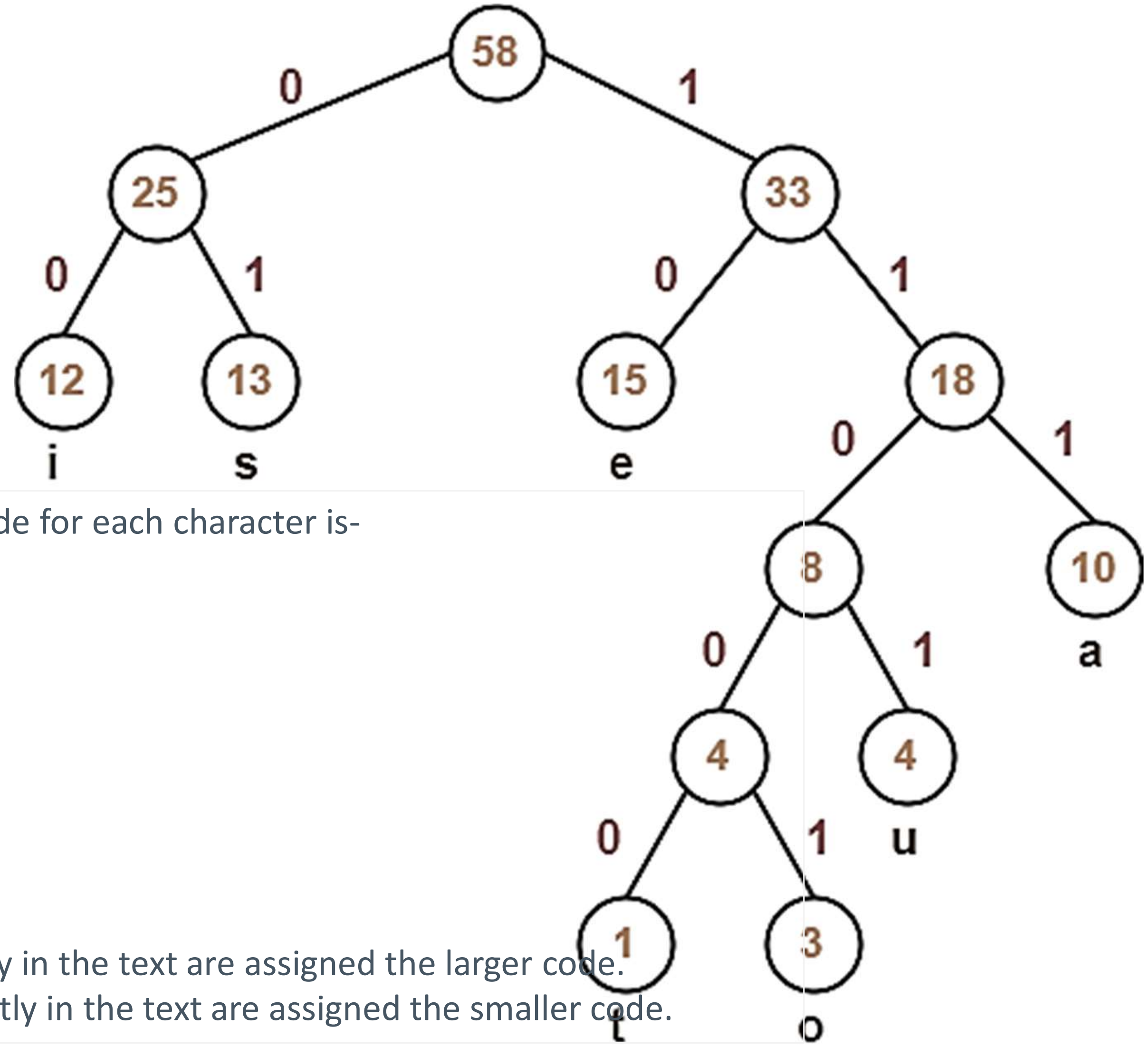
Huffman Tree

Rule

- If you assign weight '0' to the left edges, then assign weight '1' to the right edges.
- If you assign weight '1' to the left edges, then assign weight '0' to the right edges.
- After assigning weight to all the edges, the modified Huffman Tree is- NEXT SLIDE

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

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.

Average Code Length

› Using (1), we have Average code length

$= \sum (\text{ frequency}_i \text{ x code length}_i) / \sum (\text{ frequency}_i)$

= { (10 x 3) + (15 x 2) + (12 x 2) + (3 x 5) + (4 x 4) + (13 x 2) + (1 x 5) } / (10 + 15 + 12 + 3 + 4 + 13 + 1)

= 2.52

Length of Huffman Encoded Message

- Using (2), we have 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

Example

- Build the Huffman coding tree for the message

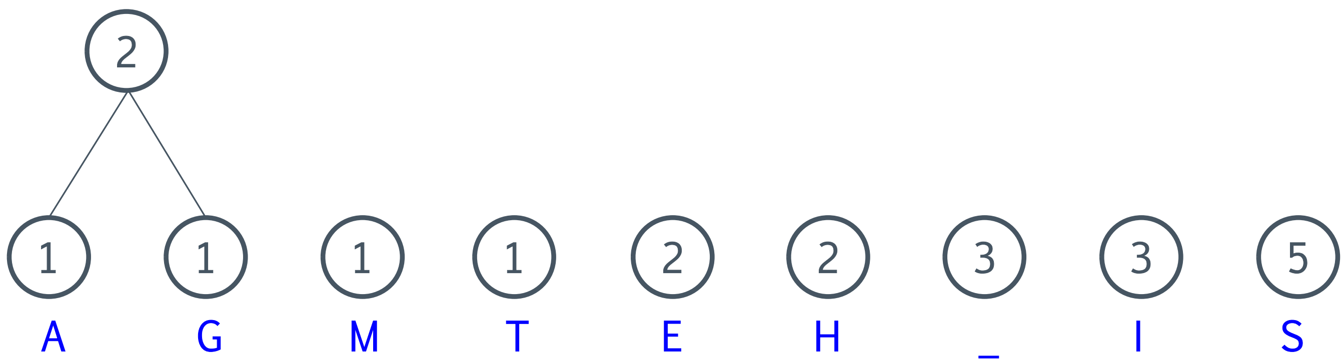
  This is his message

- Character frequencies

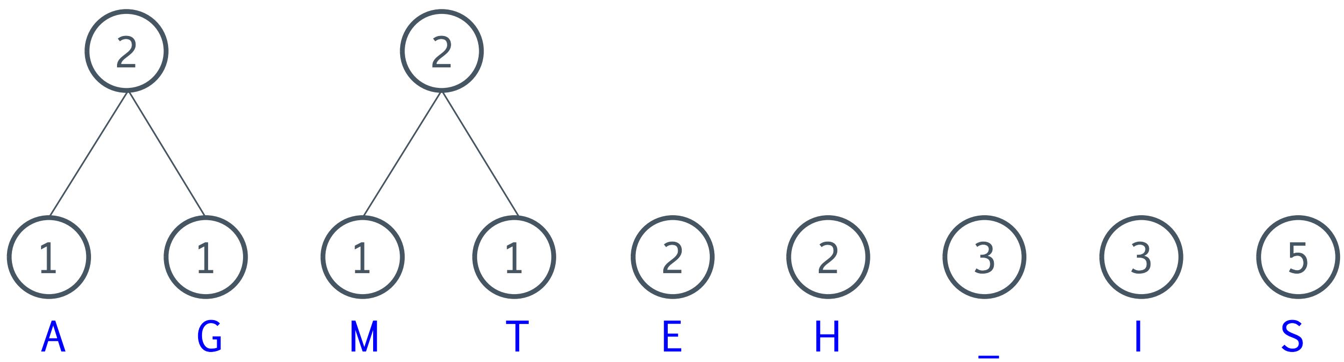
A	G	M	T	E	H	_	I	S
1	1	1	1	2	2	3	3	5

- Begin with forest of single trees

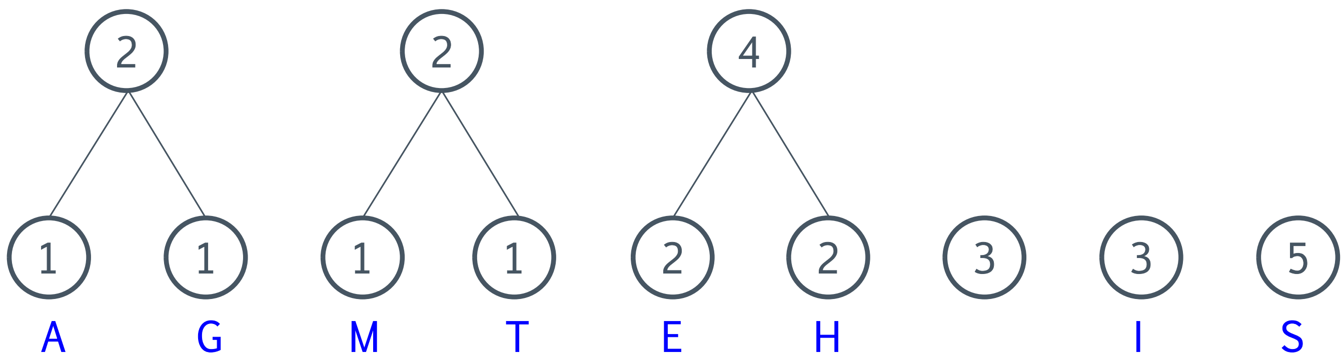
Step 1



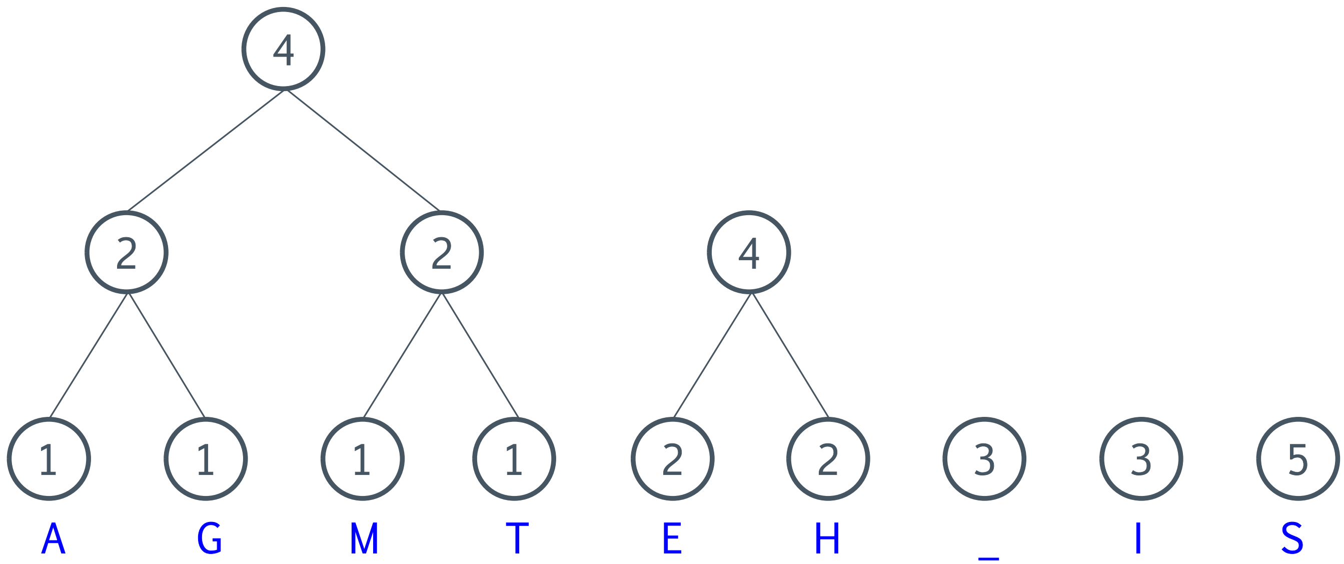
Step 2



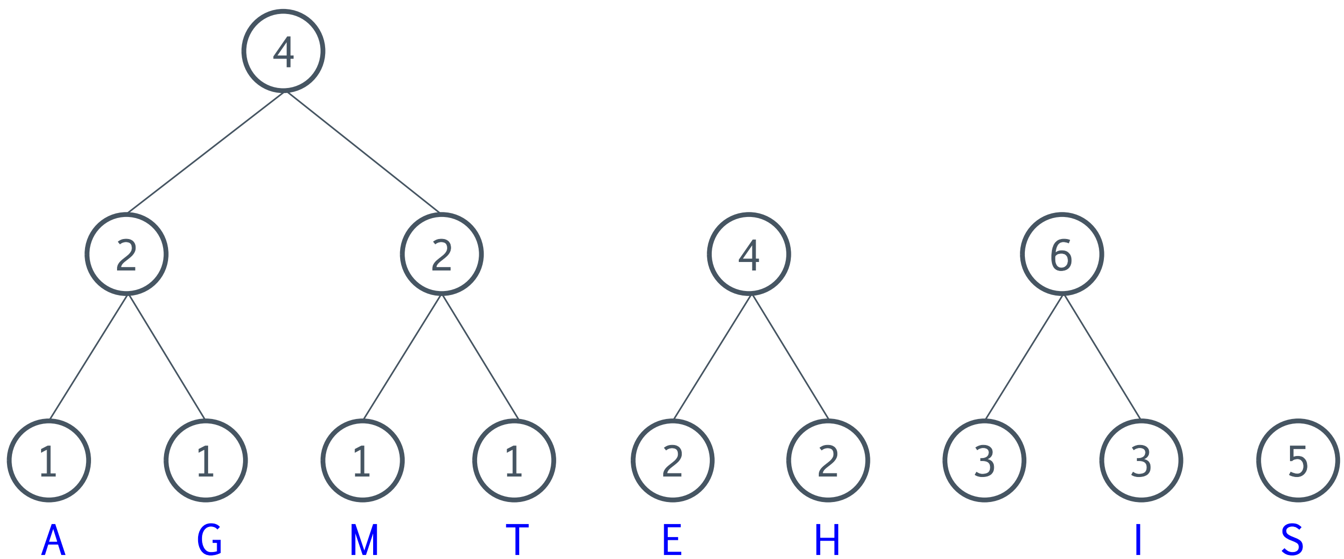
Step 3



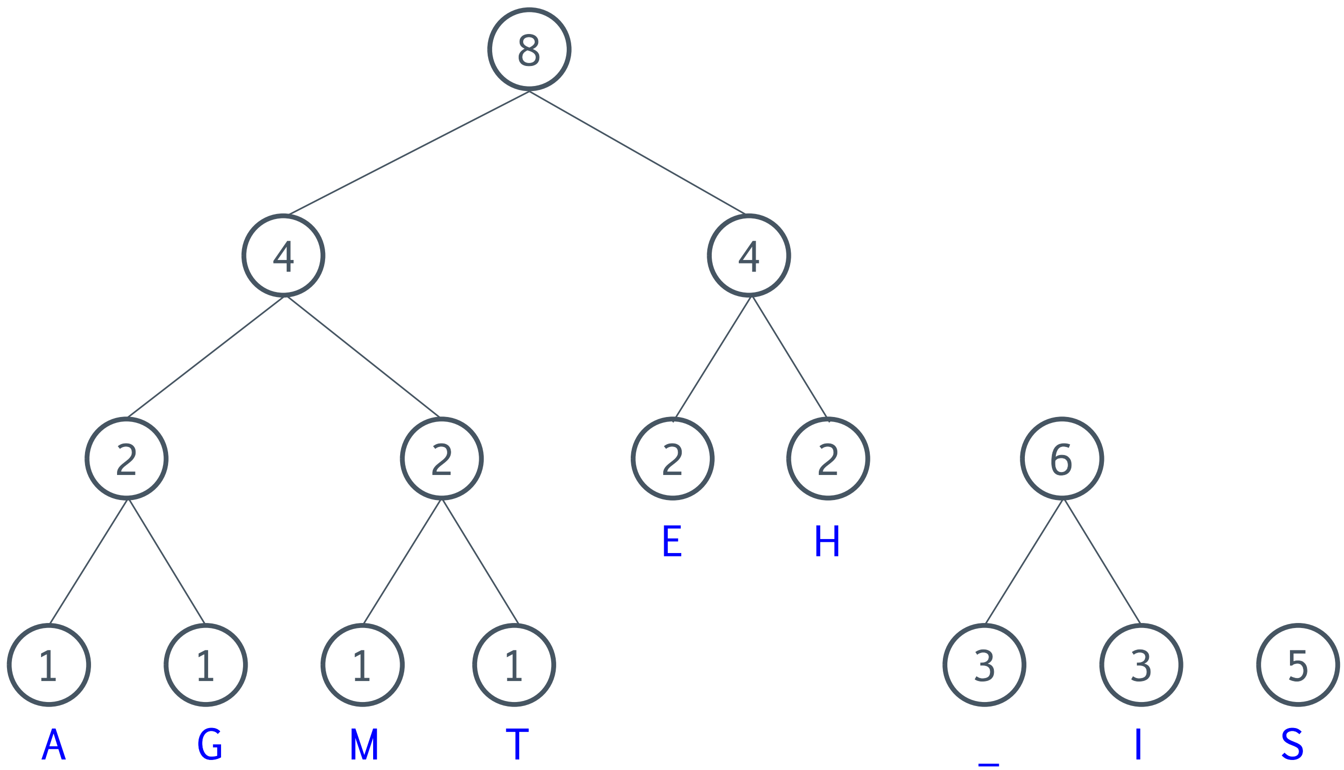
Step 4



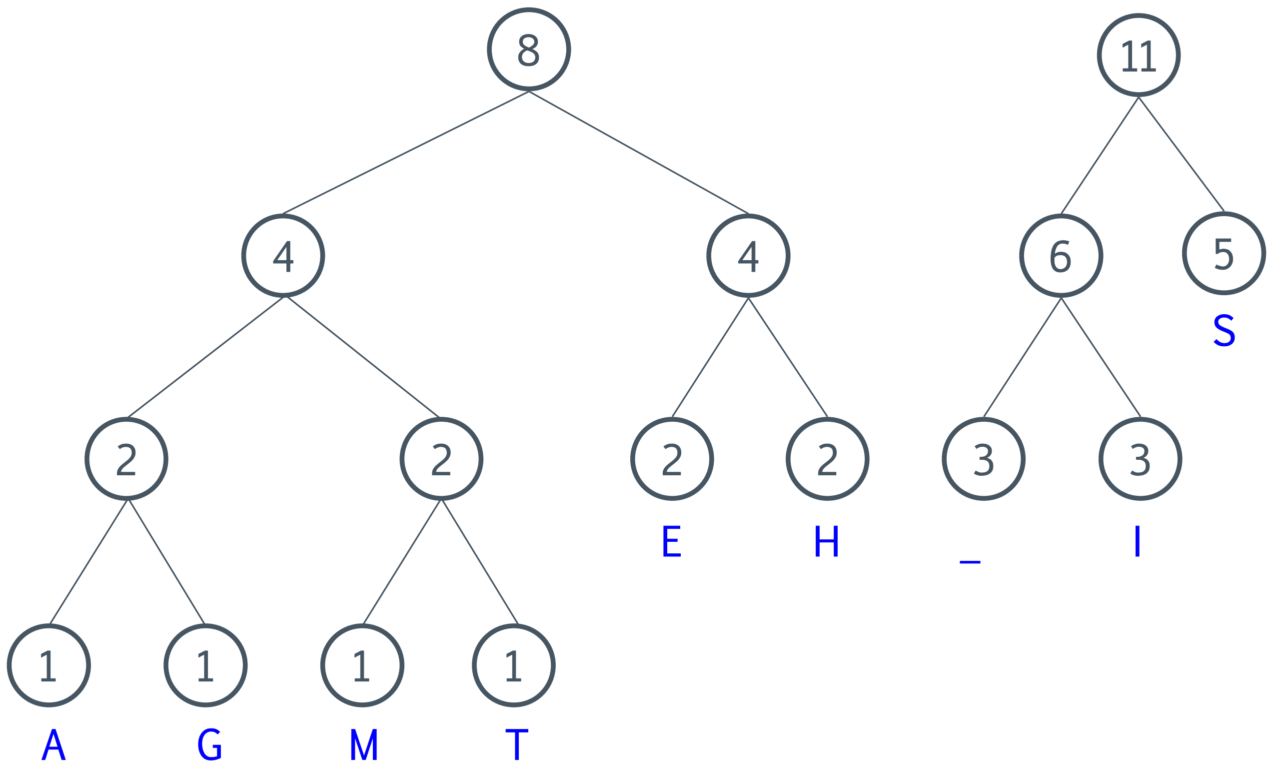
Step 5



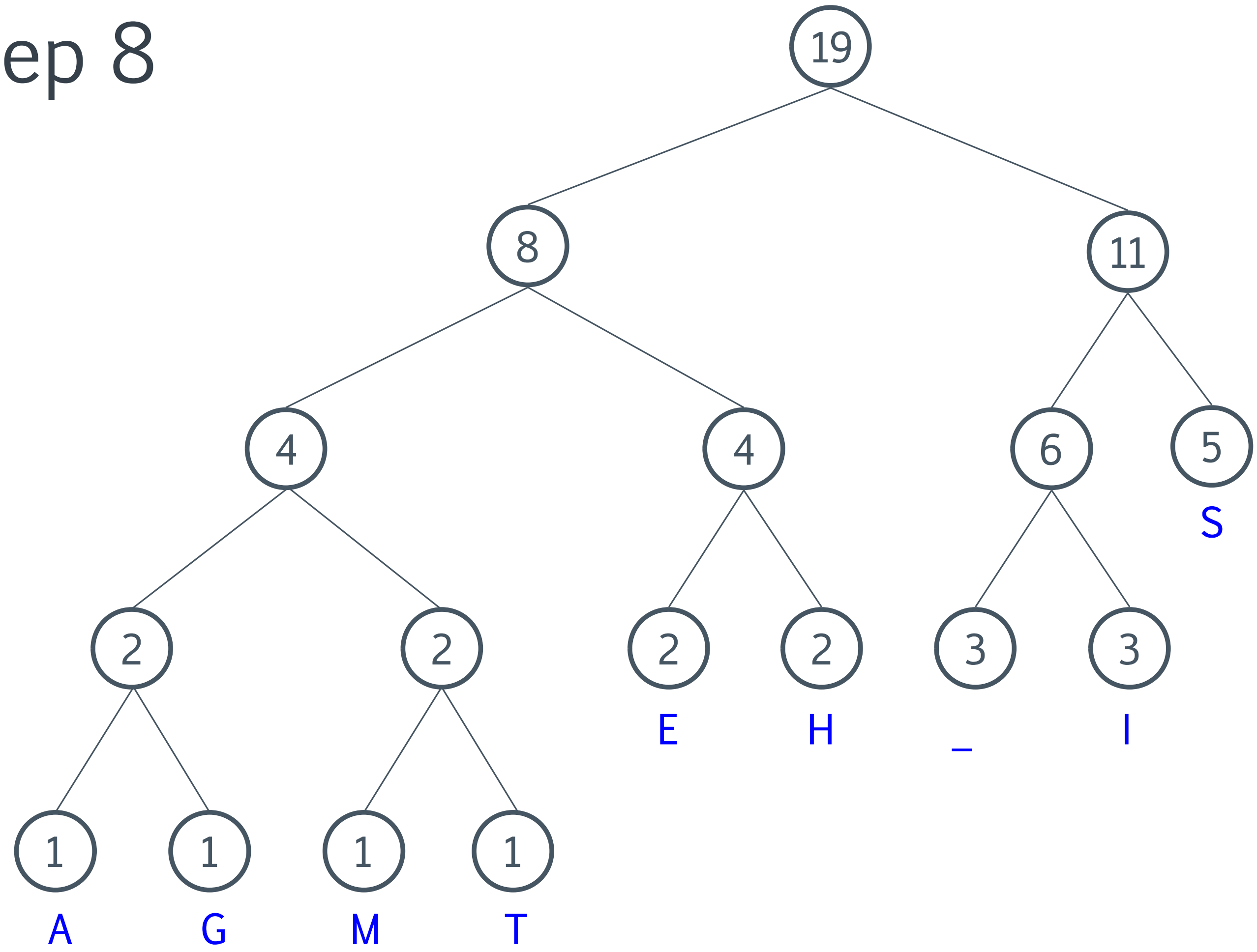
Step 6



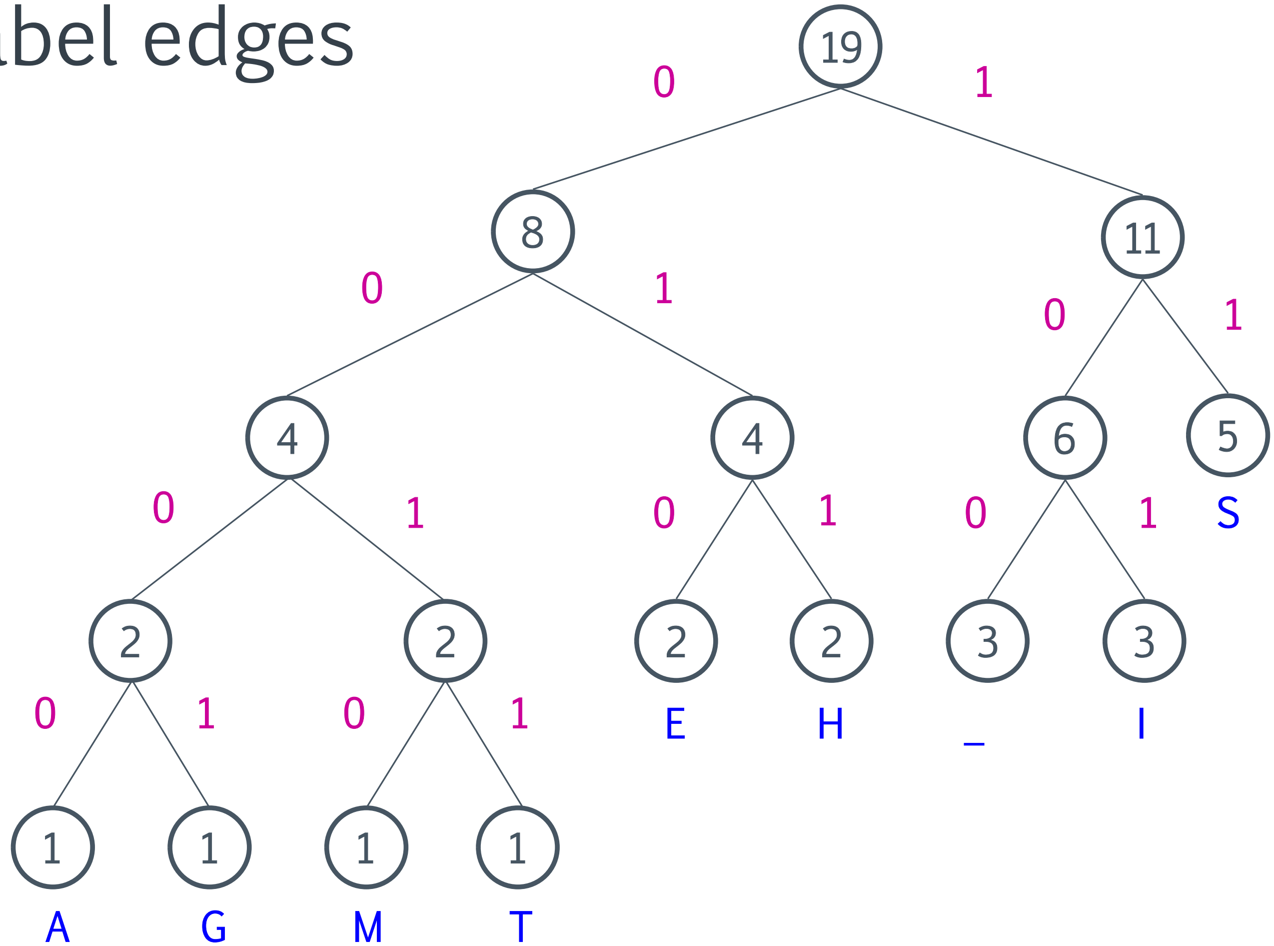
Step 7



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

001101110111100101111000111011110000100101111100000001010

Example based Simulation

- Message- BCCABBDDAECCBBAEDDCC

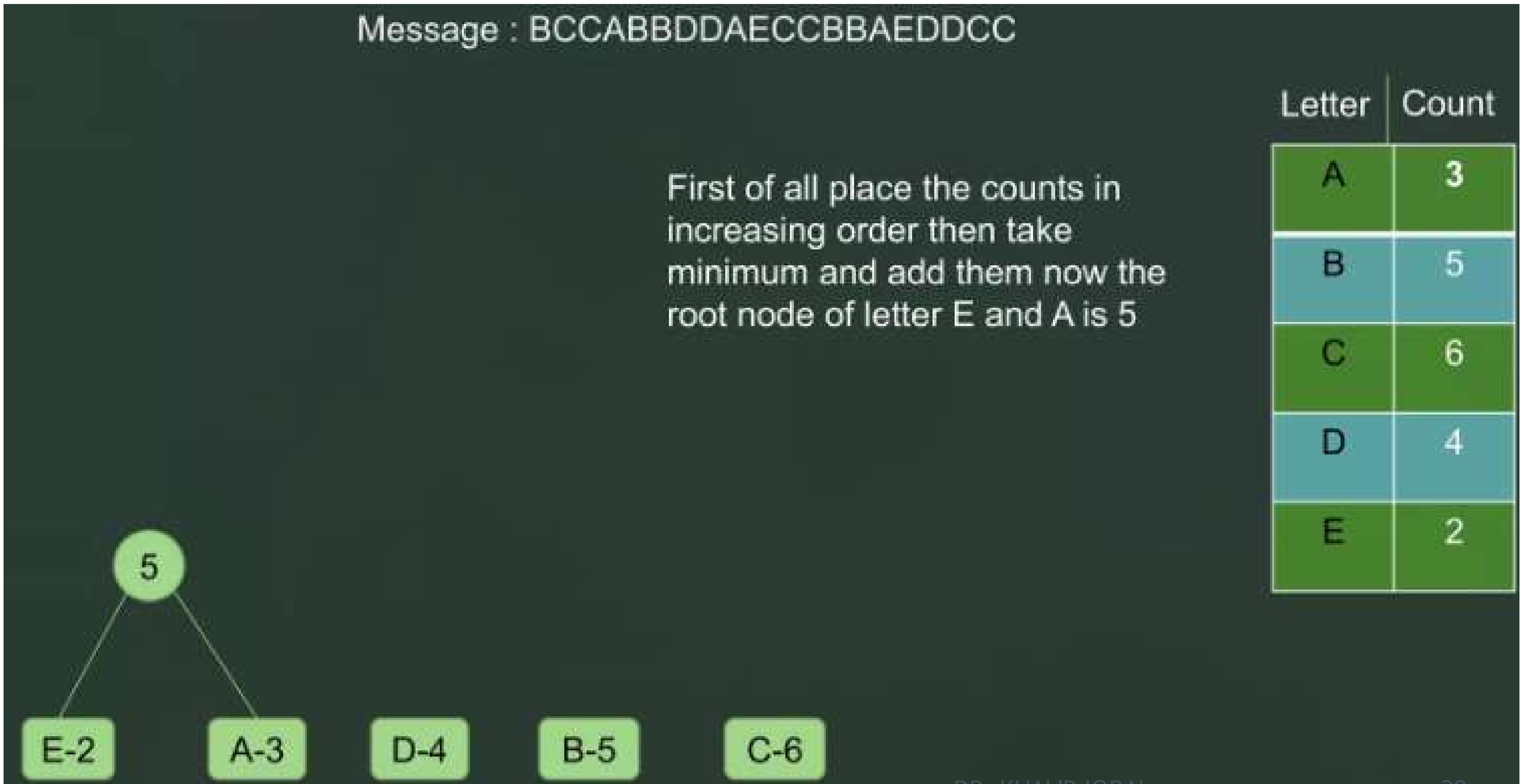
Length-20

Letter	ASCII Code	Binary Form
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101

In Electric components the alphabet is sent through ASCII code . The ASCII code letter capital A is 65 and we need 8 bis binary to convert 65.

- For 1 Letter We need 8 bits
- For 20 Letters We need 8×20=160 bits

Simulation


Message : BCCABBDDAECCBBAEDDCC
Letter
Count
A
3
B
5
C
6
D
4
E
2
First of all place the counts in increasing order then take minimum and add them now the root node of letter E and A is 5
5
E-2
A-3
D-4
B-5
C-6

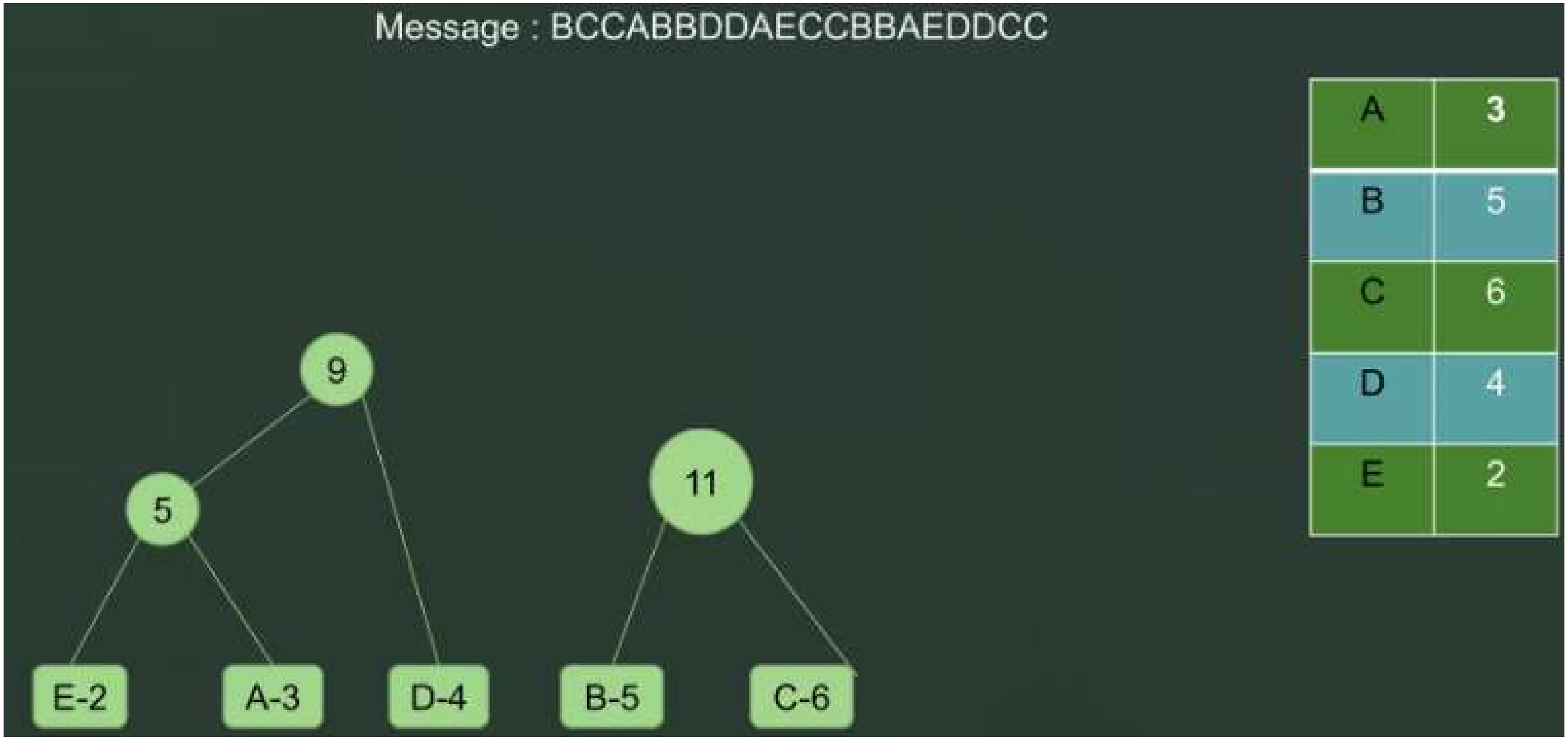
Simulation

Message : BCCABBDDAECCBBAEDDCC

Then between the root node 5 and counts take the minimum and add them up . Here 5 and 4 are minimum so we add them and make 9 as root node so continue the process.

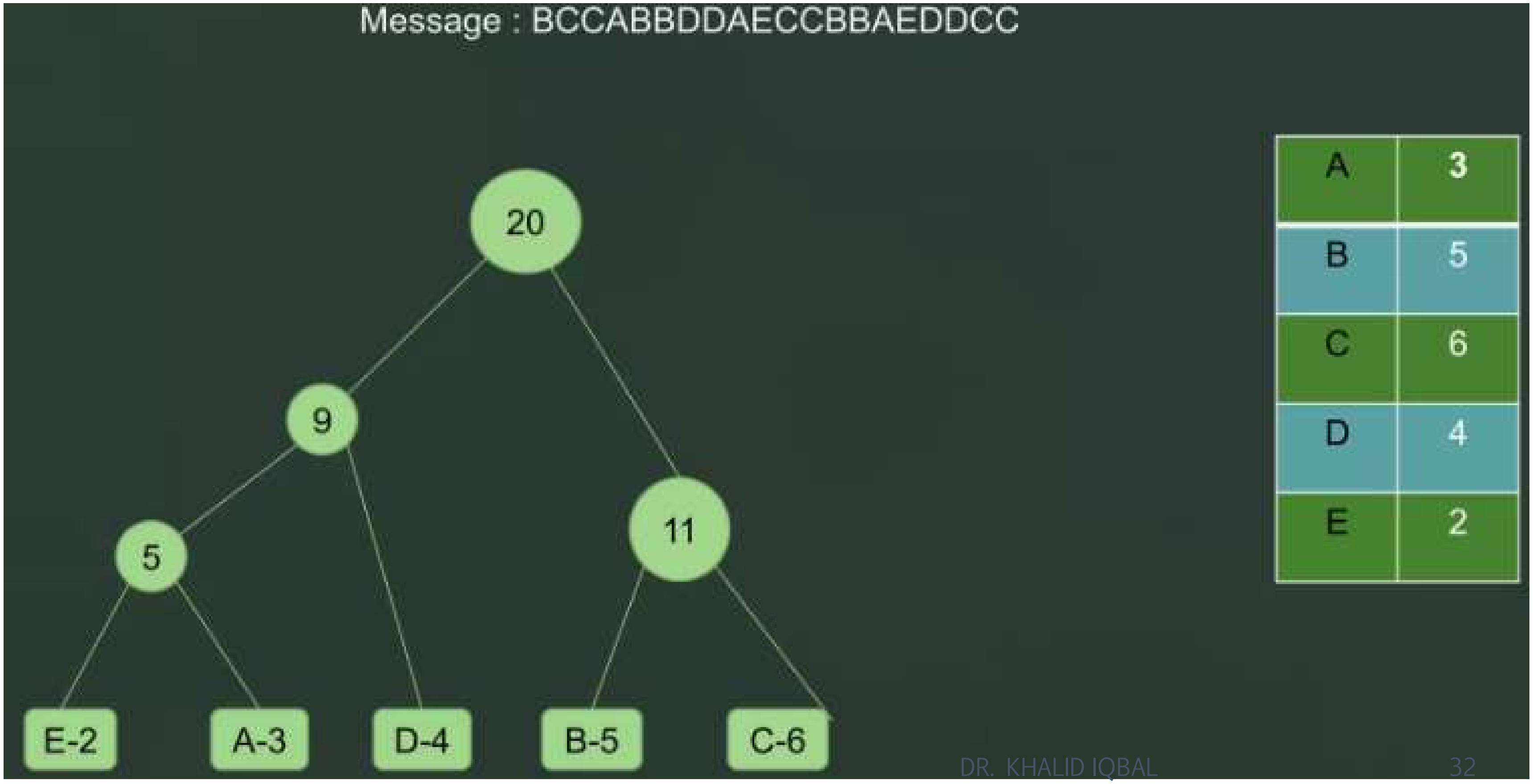
A	3
B	5
C	6
D	4
E	2

Simulation



Message : BCCABBDDAECCBBAEDDCC
9
5
11
E-2
A-3
D-4
B-5
C-6
A
3
B
5
C
6
D
4
E
2

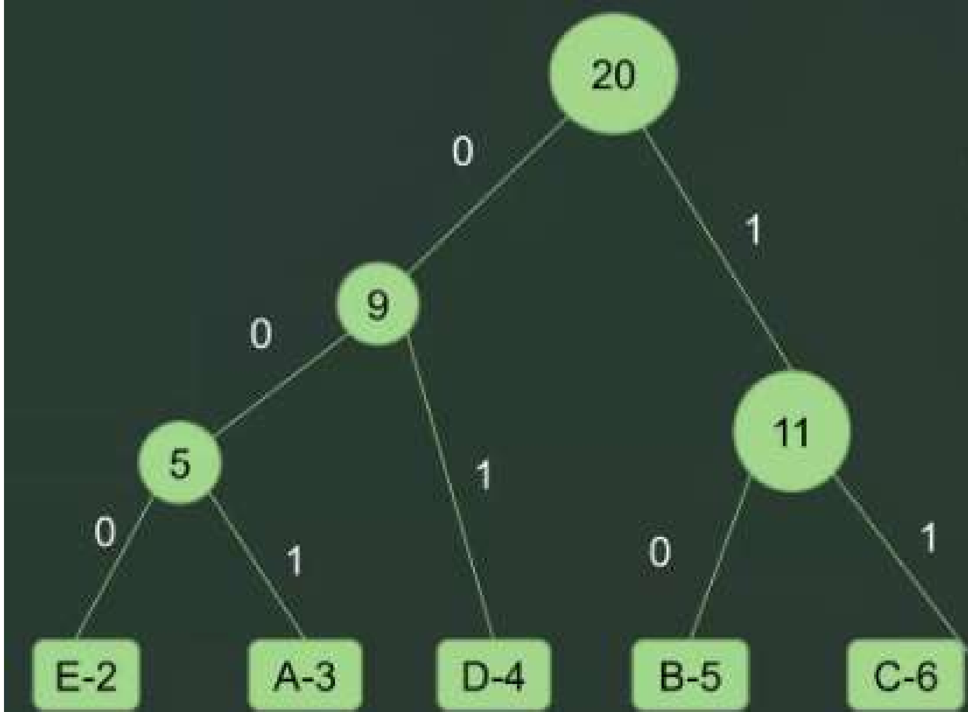
Simulation



Message : BCCABBDDAECCBBAEDDCC
20
9
11
5
E-2
A-3
D-4
B-5
C-6
A 3
B 5
C 6
D 4
E 2

Simulation

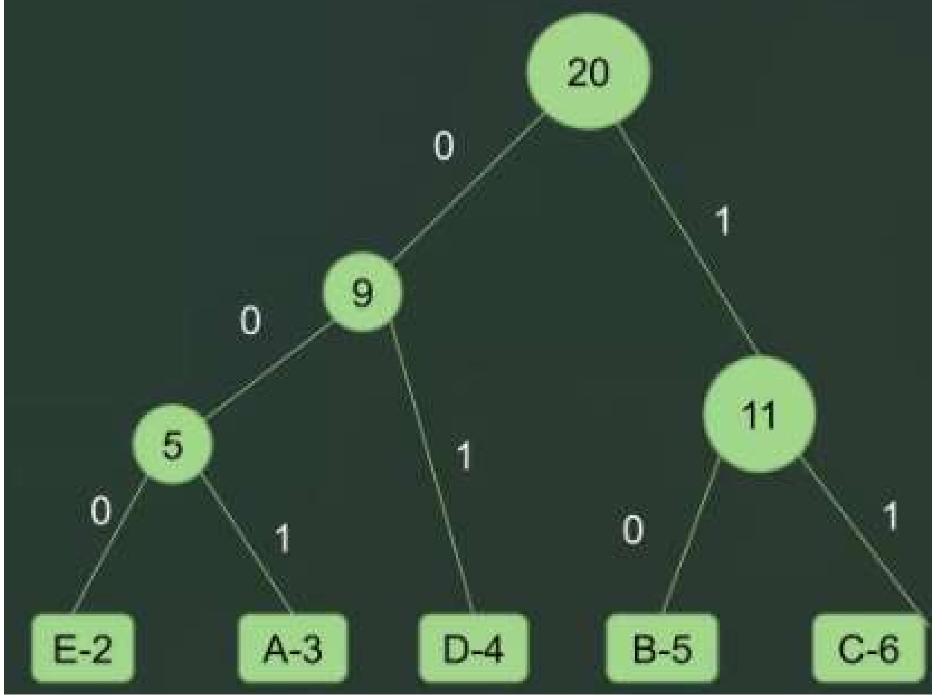
Message : BCCABBDDAECCBBAEDDCC



Mark the left hand edges as 0 and right hand edges as 1 and then traverse from root node to any letter . Suppose we want to go A from root node so the distance will be 001, for B 10 and so on.

Simulation

Message : BCCABBDDAECCBBAEDDCC



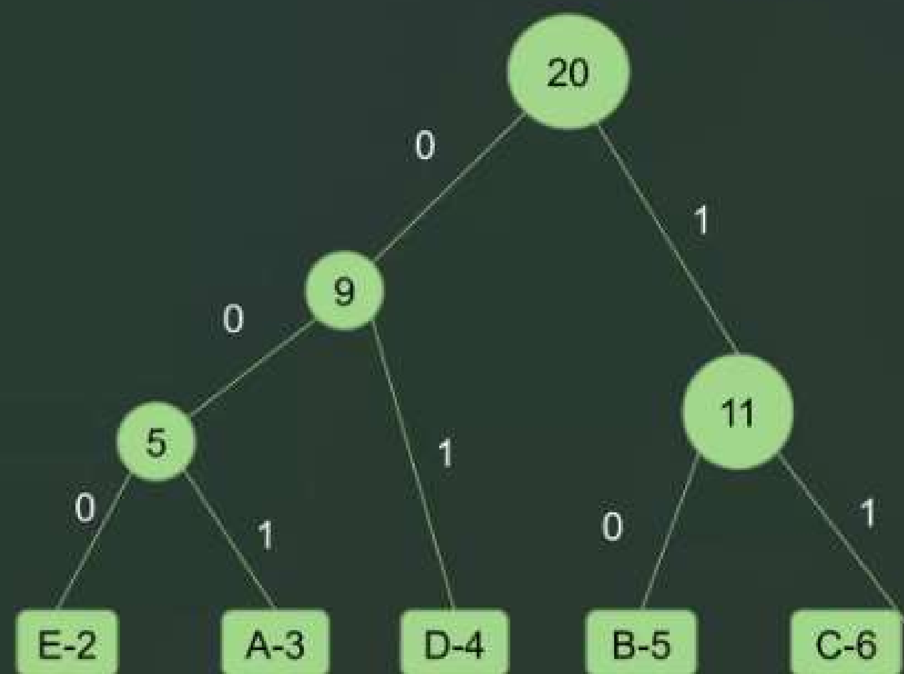
For A we need 3 bits ,B 2 bits , C 2 bits , D 2 bits, E 3 bits

For A,
Count =3
So total bits 3*3=9 bits
And so on

A	001
B	10
C	11
D	01
E	000

Simulation

Message : BCCABBDDAECCBBAEDDCC



A	3	001	3×3=9
B	5	10	5×2=10
C	6	11	6×2=12
D	4	01	4×2=8
E	2	000	2×3=6

Total=45 bits

As we see first we do need 160 bits and now we need 45 bits now we have compressed the cost and size.

Algorithm

procedure Huffman(X, $f(.)$) X is the set of symbols, whose frequencies are known in advance

$n = |X|$, the number of characters

for all $x \in X$, enqueue$((x, f(x)), Q)$ Q is a min-priority queue, implemented as binary-heap

for $i = 1$ to n -1

allocate a new tree node z

$left\_child$ =deletemin(Q)

$right\_child$ =deletemin(Q)

$f(z) = f(left\_child) + f(right\_child)$

Make $left\_child$ and $right\_child$ the children of z

enqueue$((z, f(z)), Q)$

Complexity of Algorithm

procedure Huffman(X, $f(.)$) Thus, the algorithm needs O(nlogn)
$n = |X|$, the number of characters
for all $x \in X$, enqueue($(x, f(x)), Q$) needs O(nlogn)
for $i = 1$ to n -1 Thus, the loop needs O(nlogn)
allocate a new tree node z
$left\_child$ =deletemin(Q) needs O(logn)
$right\_child$ =deletemin(Q) needs O(logn)
$f(z) = f(left\_child) + f(right\_child)$
Make $left\_child$ and $right\_child$ the children of z
enqueue($(z, f(z)), Q$) needs O(logn)

Pseudo-code

Time complexity of Huffman Coding is O(nlogn) ,where n is the number of unique characters.

```
Huffman (C)
  n = |C|
  Q = C
  for i = 1 to n-1
     allocate new node z
     z.left = x = Extract-min (Q)
     z.right = y = Extract-min (Q)
     insert (Q, z)
  return Extract-Min (Q)  // returns the root
```

$\boxed{\text{Complexity} = O(n \lg n)}$

Application of Huffman Encoding

Generic File Compression:

- Files: GZIP, BZIP, 7Z
- Archives : 7z
- File System : NTFS,FS+,ZFS

Multimedia :

- Image : GIF, ZPEG
- Sound : Mp3
- Video : MPEG, HDTV

Communication :

- ITU-T T4 Group 3 Fax
- V.42 Bis modem
- Skype

Databases : Google,Facebook,...

Advantages :

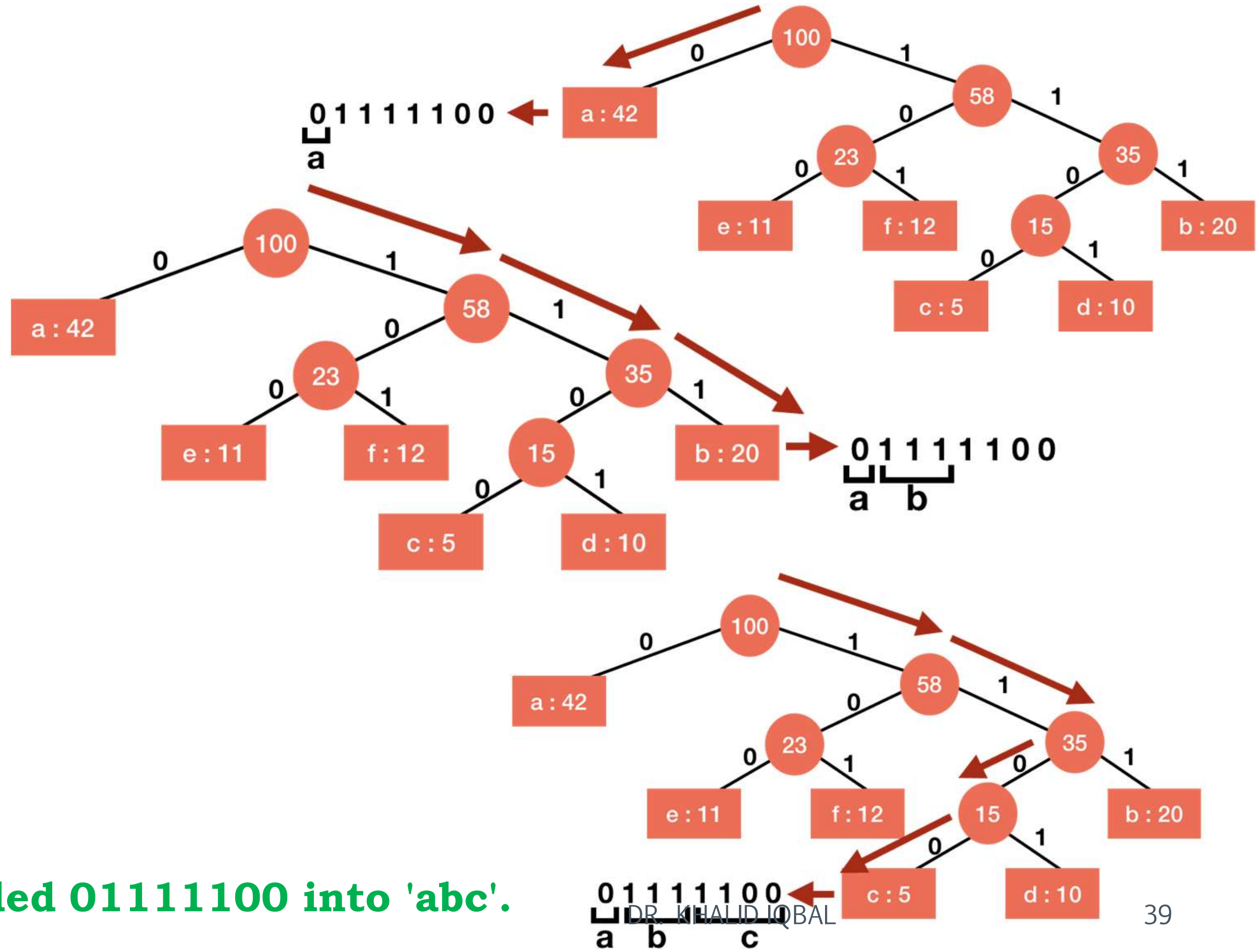
- The Huffman Coding has the minimum average length.
- Easy to implement and fast.

Disadvantages :

- Requires two passes over the two input (one to compute frequencies, one for coding),thus encoding is slow.
- Requires storing the Huffman codes(or at least character frequencies)in the encoded file, thus reducing the compression benefit obtained by encoding.

For example, let's take a case of 01111100, Decoding

Character	Frequency	Code
a	12	0
b	20	111
c	5	1100
d	10	1101
e	11	100
f	12	101



Thus, we have decoded 01111100 into 'abc'.

Activity Selection Problem (Greedy Approach)

- **Activity Selection problem** is an approach of selecting non-conflicting tasks based on start and end time and can be solved in *O(N log N)* time using a simple greedy approach.
- Modifications of this problem are complex and interesting which we will explore as well.
- With Dynamic Programming approach, the time complexity will be **O(N^3)** that is lower performance.

Problem Statement

- **Activity Selection** is that "Given a set of n activities with their start and finish times, we need to select **maximum number of non-conflicting activities** that can be performed by a single person, given that the person can handle only one activity at a time."
- The Activity Selection problem follows **Greedy approach** i.e. at every step, we can make a choice that looks best at the moment to get the optimal solution of the complete problem.

Objective

- Our objective is to complete maximum number of activities. So, choosing the activity which is going to finish first will leave us maximum time to adjust the later activities.
- This is the intuition that greedily choosing the activity with earliest finish time will give us an optimal solution.
- By induction on the number of choices made, making the greedy choice at every step produces an optimal solution, so we chose the activity which finishes first.
- If we sort elements based on their starting time, the activity with least starting time could take the maximum duration for completion, therefore we won't be able to maximise number of activities.

Applications

- Scheduling events in a room having multiple competing events
- Scheduling and manufacturing multiple products having their time of production on the same machine
- Scheduling meetings in one room
- Several use cases in Operations Research

Activity Selection Algorithm

```
Activity-Selection(Activity, start, finish)
     Sort Activity by finish times stored in finish
     Selected = {Activity[1]}
     n = Activity.length
     j = 1
     for i = 2 to n:
         if start[i] ≥ finish[j]:
             Selected = Selected U {Activity[i]}
             j = i
 return Selected
```

Time Complexity:

- When activities are sorted by their finish time: O(N)
- When activities are not sorted by their finish time, the time complexity is O(N log N) due to complexity of sorting

Example

	0	1	2	3	4	5	6
START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

SELECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[1]>=END[0], SELECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[2]<END[1], REJECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[3]<END[1], REJECTED

```
# Python Code for Activity Selection
# Function for Activity Selection
def ActivitySelection(start, finish, n):
    print("The following activities are selected:");
    j = 0
    print(j,end=" ")
    for i in range(1,n):
        if start[i] >= finish[j]:
            print(i,end=" ")
            j = i
# Driver Code
start = [1, 3, 2, 0, 5, 8, 11]
finish = [3, 4, 5, 7, 9, 10, 12]
n = len(start)
ActivitySelection(start, finish, n)
# Output
# The following activities are selected:
# 0 1 4 6
```

Example

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[4] >=END[2], SELECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[5]<END[4], REJECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[6]>=END[4], SELECTED

```python
# Python Code for Activity Selection
# Function for Activity Selection
def ActivitySelection(start, finish, n):
    print("The following activities are selected:");
    j = 0
    print(j,end=" ")
    for i in range(1,n):
        if start[i] >= finish[j]:
            print(i,end=" ")
            j = i
# Driver Code
start = [1, 3, 2, 0, 5, 8, 11]
finish = [3, 4, 5, 7, 9, 10, 12]
n = len(start)
ActivitySelection(start, finish, n)
# Output
# The following activities are selected:
# 0 1 4 6
```

Thank You!!!

Have a good day