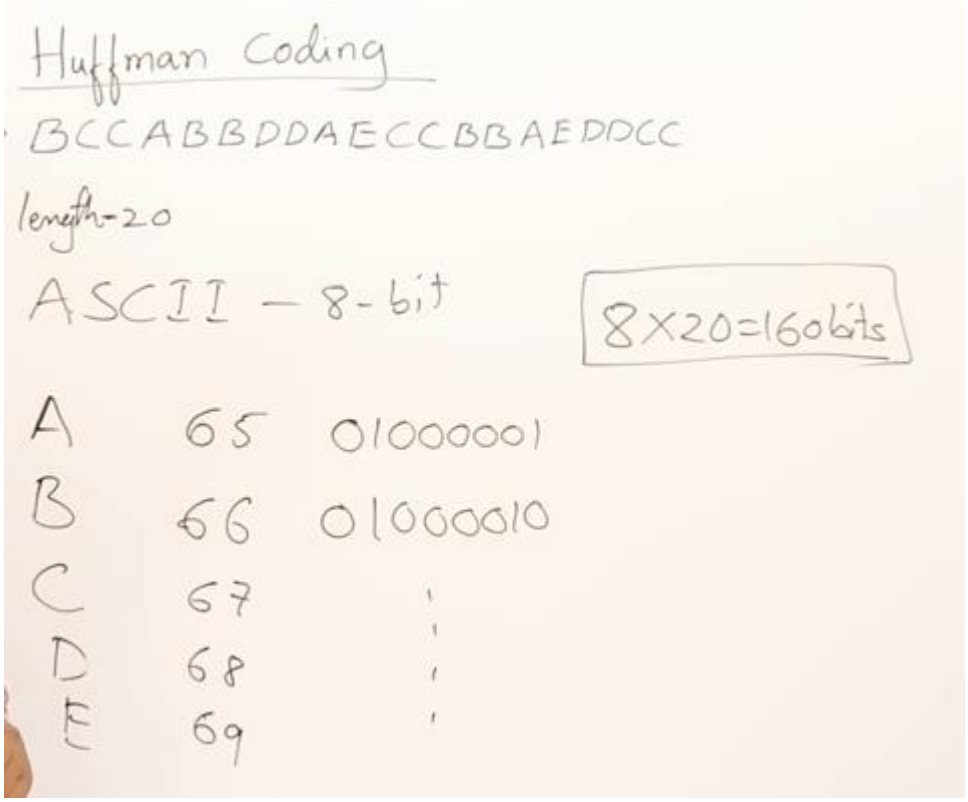
Huffman coding is a lossless data compression algorithm. In this algorithm, a variable-length code is assigned to input different characters. The code length is related to how frequently characters are used. Most frequent characters have the smallest codes and longer codes for least frequent characters.

There are mainly two parts. First one to create a Huffman tree, and another one to traverse the tree to find codes.

- 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 Coding

Message → BCCABBDDAECCBBAEDDCC

character	count/frequency		Code
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	011
E	2	2/20	100

0/1 bit

0 0 bit
0 1
1 0
1 1

--- bit
0
1
7

Huffman Coding

Message → BCCABBDDAECCBBAEDDCC

001 010 . . .

character	count/frequency		Code
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	011
E	2	2/20	100
	20		

20×3 = 60 bits

5×8 bit (↑ character) 5×3 (↑ codes)

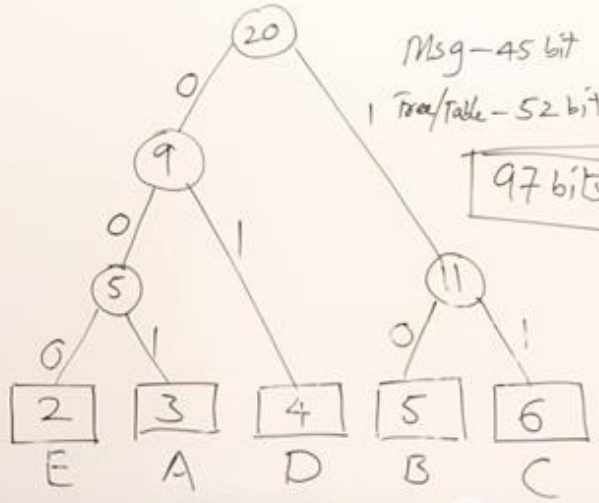
40 + 15 = 55

Msg — 60 bits
Table — 55 bits
115 bits

Huffman Coding

Message → BCCABBDDAECCBBAEDDCC

10 11 11 001 10 10 01 01 - - -



Msg – 45 bit

Tree/Table – 52 bits

97 bits

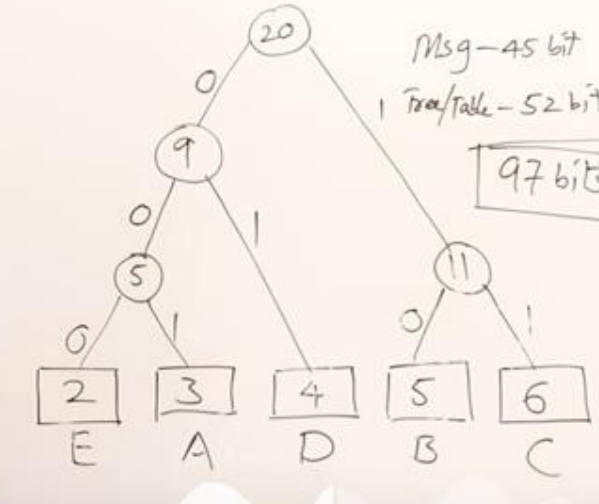
Char	Count	Code	
A	3	001	3×3=9
B	5	10	5×2=10
C	6	11	6×2=12
D	4	01	4×2=8
E	2	000	2×3=6
	20	12 bit	45 bit

5×8 bit = 40 bits +

Huffman Coding

Message → BCCABBDDAECCBBAEDDCC

10 11 11 001 10 10 01 01 - - -



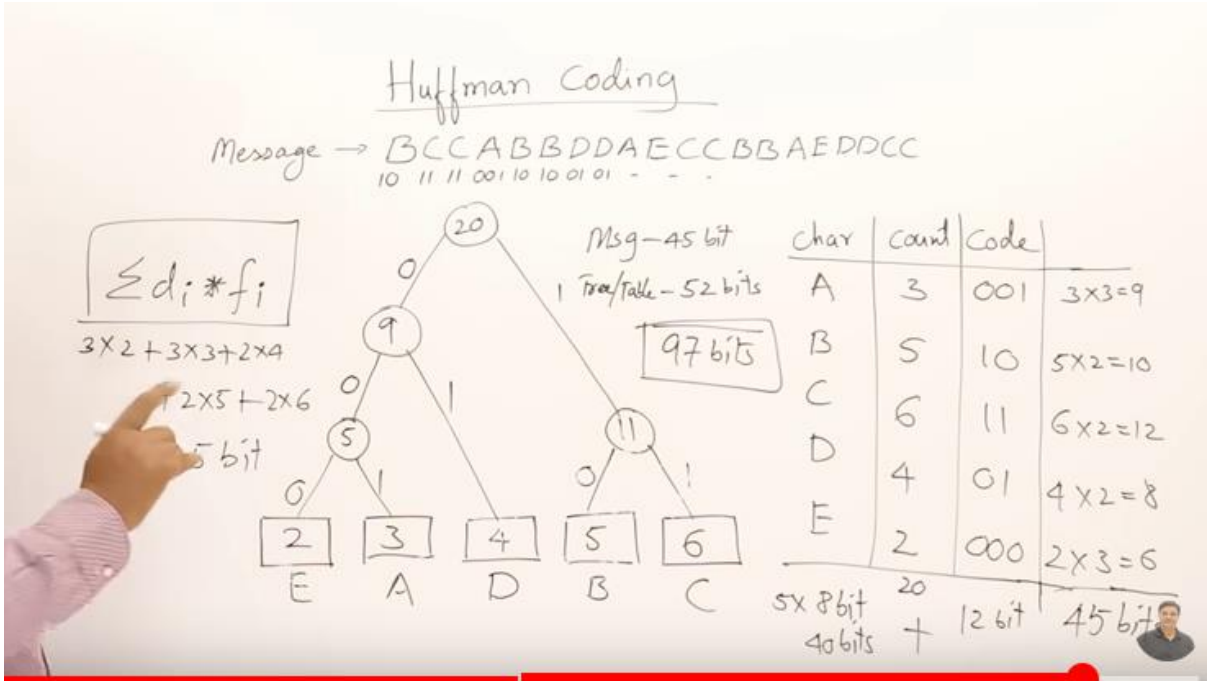
Msg – 45 bit

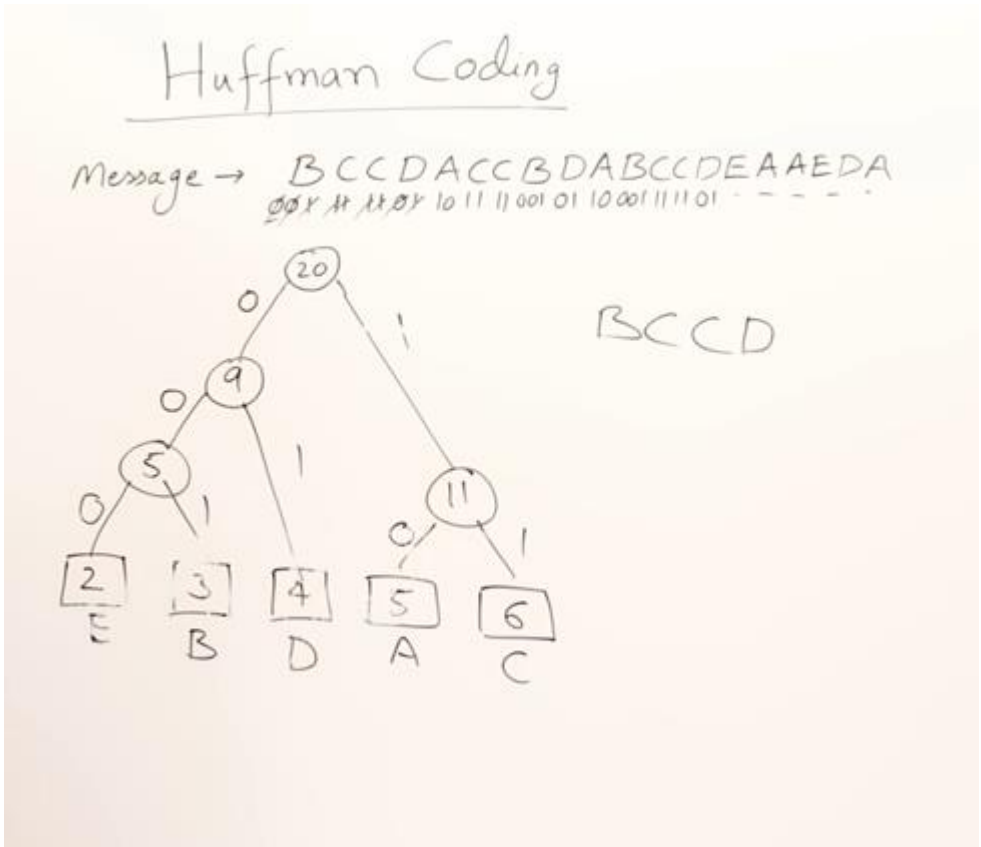
Tree/Table – 52 bits

97 bits

Char	Count	Code	
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
	20	12 bit	45 bit

5×8 bit = 40 bits +


Huffman Coding
Message → BCCABBDDAECCBBAEDDCC
10 11 11 001 10 10 01 01 - - -
Σdi*fi
3X2+3X3+2X4
+2X5+2X6
bit
20
0
9
1
0
5
1
11
0
1
0
1
2
3
4
5
6
E
A
D
B
C
Msg–45 bit
Tree/Table–52 bits
97 bits
char | count | Code
A 3 001 3X3=9
B 5 10 5X2=10
C 6 11 6X2=12
D 4 01 4X2=8
E 2 000 2X3=6
5X 8bit 40 bits
20 + 12 bit
45 bit


Huffman Coding
Message → BCCDACCBDABCCDEAAEDA
10 11 11 001 01 10 000 1111 01 - - - -
20
0
9
1
0
5
1
11
0
1
0
1
2
3
4
5
6
E
B
D
A
C
BCCD

Example 2

Huffman Coding

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.

How Huffman Coding works?

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

Initial string

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.

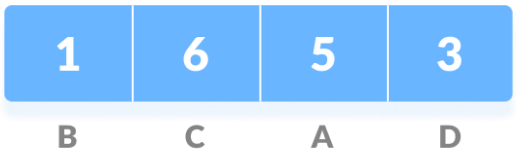
Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.

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.

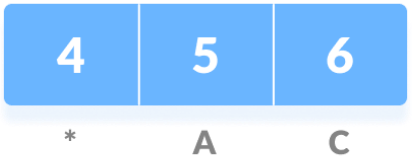


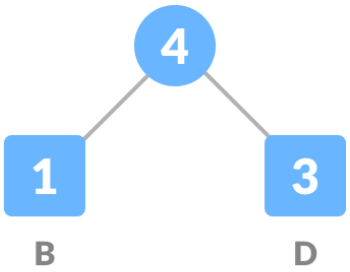
Frequency of string

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

queue *Q*. Characters sorted according to the frequency

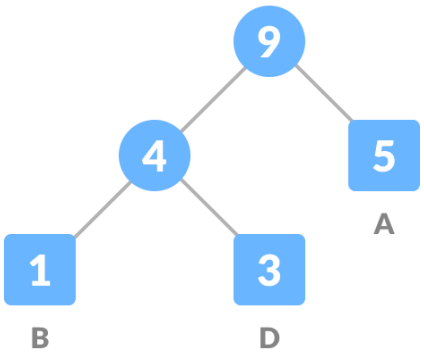
3. Make each unique character as a leaf node.
4. Create an empty node *z*. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of *z*. Set the value of the *z* as the sum of the above two minimum frequencies.



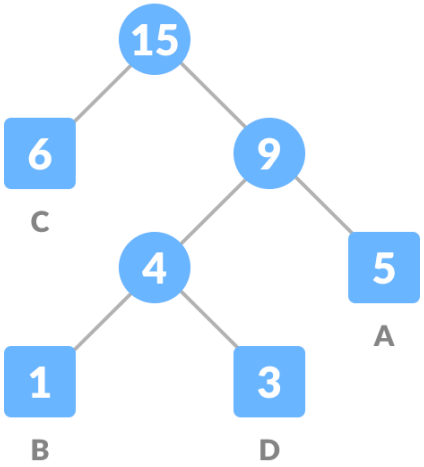


Getting the sum of the least numbers

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.

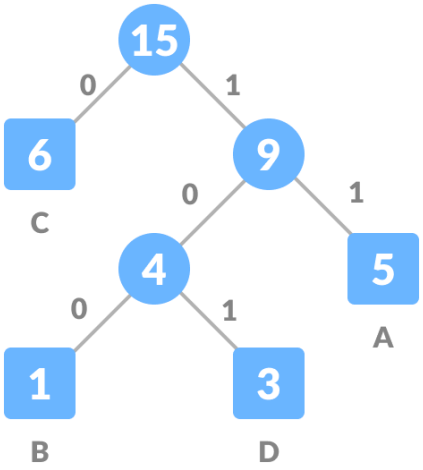


7. Repeat steps 3 to 5 for all the characters.



Repeat steps 3 to 5 for all the characters.
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.



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.

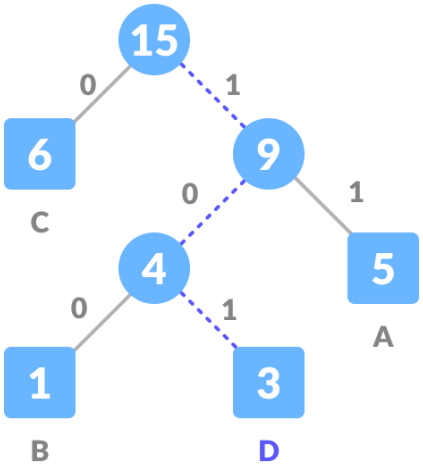
Character	Frequency	Code	Size
A	5	11	5*2 = 10
B	1	100	1*3 = 3
C	6	0	6*1 = 6
D	3	101	3*3 = 9
4 * 8 = 32 bits	15 bits		28 bits

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

Decoding the code

For decoding the code, we can take the code and traverse through the tree to find the character.

Let 101 is to be decoded, we can traverse from the root as in the figure below.



Decoding