HUFFMAN TREE

PRESENTATION

The Problem

Huffman Coding: is a technique of compressing data to reduce its size without losing any of the details. Huffman Coding is generally useful to **compress the data** in which there are frequently occurring characters.

BCAADDDCCACACAC

ASCII - 8 Bits

8*15=120

so, 120 bits are required to send this string

The Problem

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 Strategy

1-Calculate the frequency of each characrer in a string .

BCAADDDCCACACAC

CHARACTER	COUNT/FREQUENCY
A	5
B	1
C	6
D	3

The Strategy

2-Sort the characters in increasing order of the frequency(Priority)

CHARACTER	COUNT / FREQUENCY
B	1
D	3
A	5
C	6

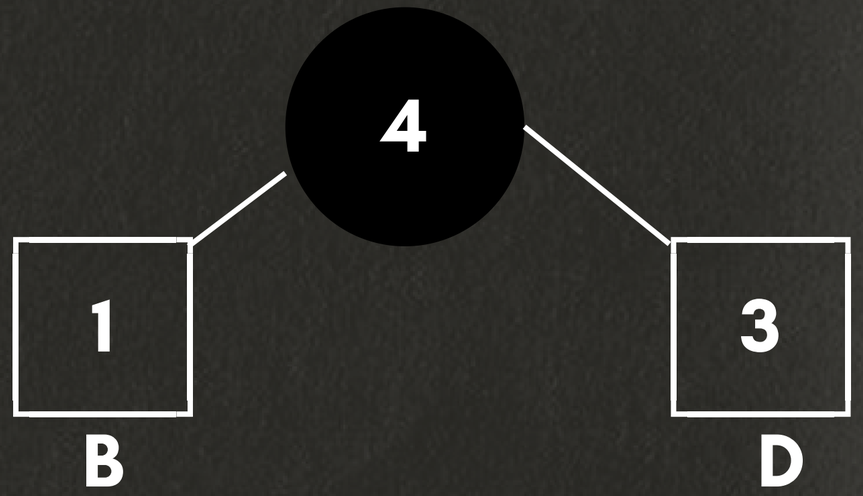
The Strategy

3-Make each unique character as a leaf node.

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.

~~B-1 , D-3~~, A-5 , C-6

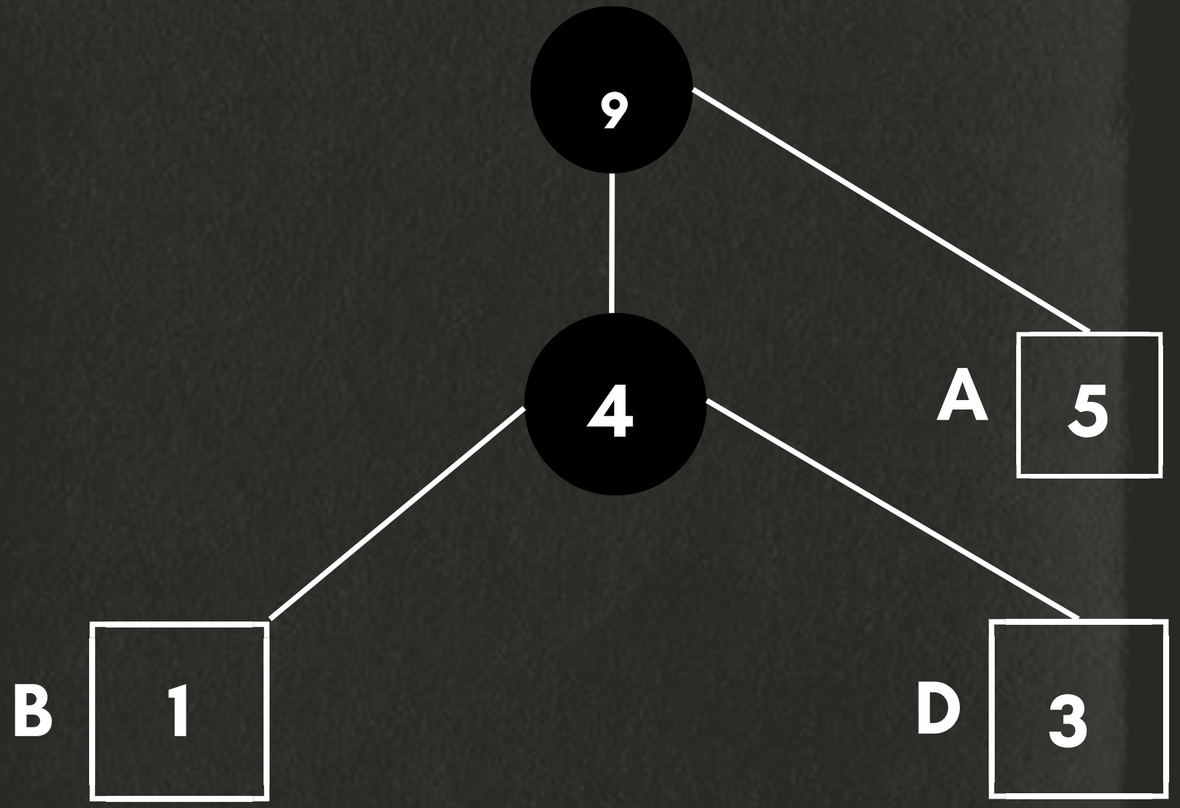
4



The Strategy

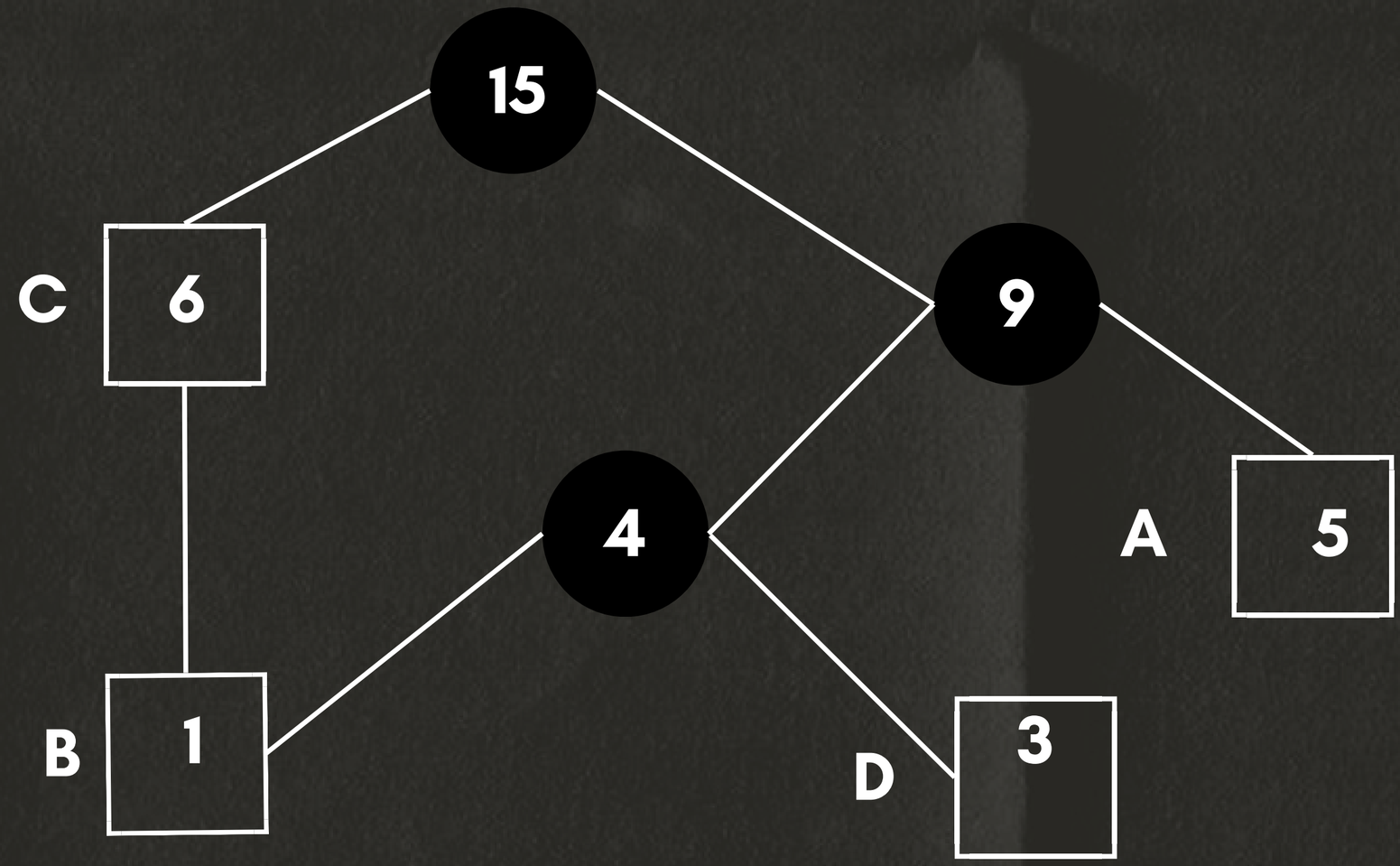
4- Remove these two minimum frequencies from Q and add the sum into the list of frequencies then Insert node z into the tree.

~~A-5~~ , C-6, 9



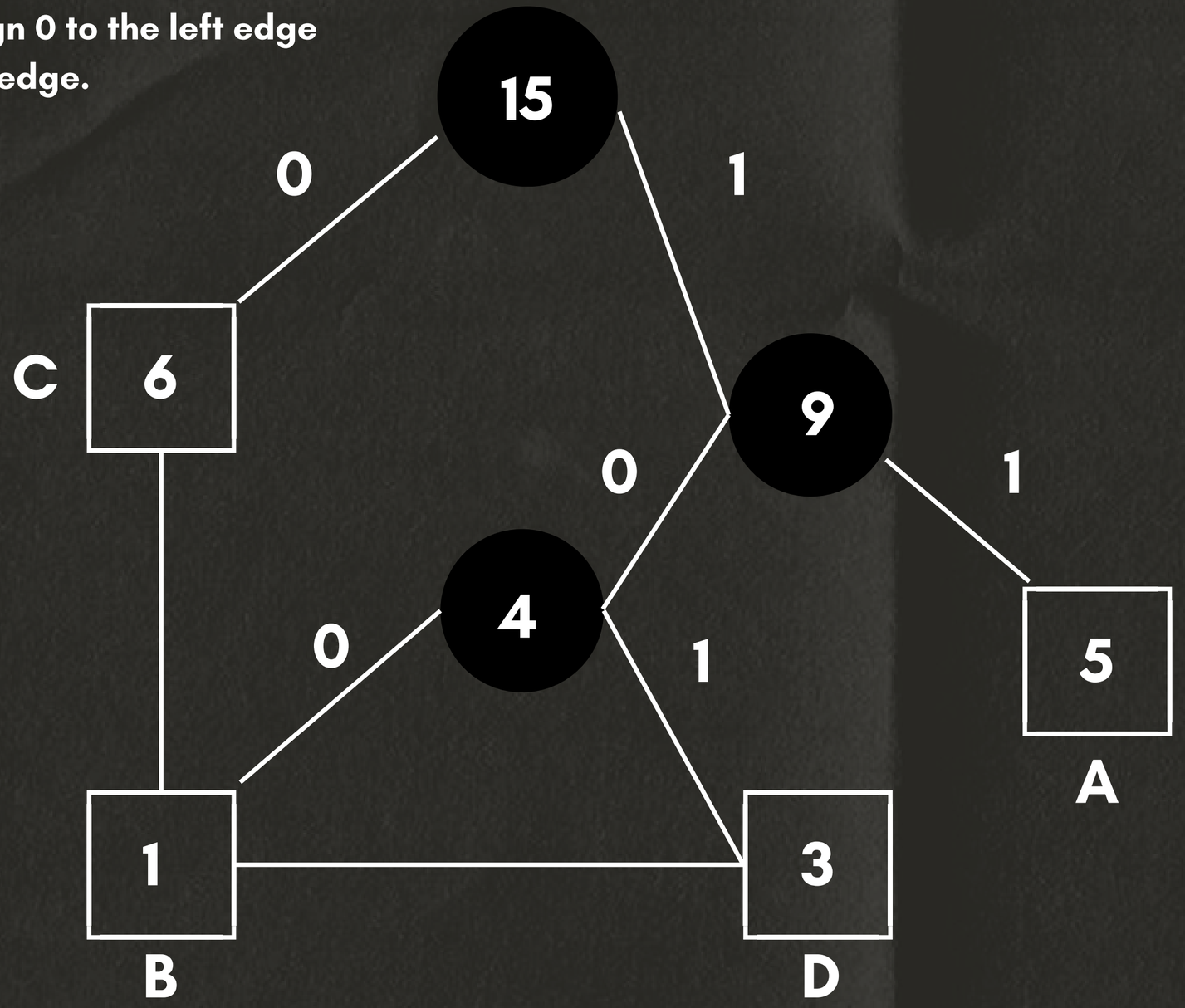
The Strategy

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



The Strategy

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



The Strategy

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.

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

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

The Algorithm

Create a Priority queue Q consisting of each unique character.
sort then in ascending order of their frequencies.
for all the unique characters:
create a newNode
extract minimum value from Q and assign it to leftChild of newNode
extract minimum value from Q and assign it to rightChild of newNode
calculate the sum of these two minimum values and assign it to the value of newNode
insert this newNode into the tree
return rootNode

The Analysis

Best Case:

- In the best case scenario where all characters have equal frequencies, while building the Huffman tree, combining nodes might be simpler due to the balanced nature of the tree.
- However, despite the simplicity in tree construction, the time complexity remains O(n log n) due to the sorting of frequencies and constructing the tree.

The Analysis

Worst Case:

- In the worst case scenario where characters have highly uneven frequencies, while building the Huffman tree, some characters may be deeper in the tree, necessitating longer codes and potentially resulting in a less balanced tree.
- Despite the potentially more complex tree structure, the time complexity for encoding each unique character based on its frequency still remains O(n log n).

The reason behind this uniformity in time complexity lies in the nature of the operations involved, particularly the tree construction, sorting of frequencies, and encoding characters based on this constructed tree, which are invariant regardless of the distribution of character frequencies.

Thanks For Watching

Presented by : Moustafa Mohamed