Huffman Coding

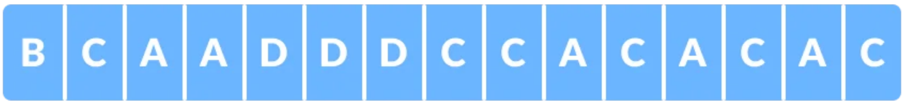
In this tutorial, you will learn how Huffman Coding works. Also, you will find working examples of Huffman Coding in C, C++, Java and Python.

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.



B C A A D D D C C A C A C A C

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.

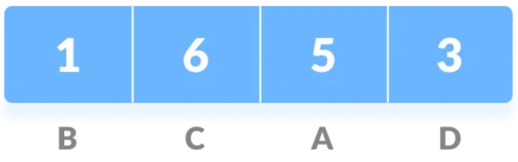
Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.

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.

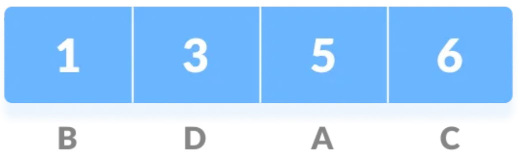
Huffman coding is done with the help of the following steps.

1. Calculate the frequency of each character in the string.



Frequency of string

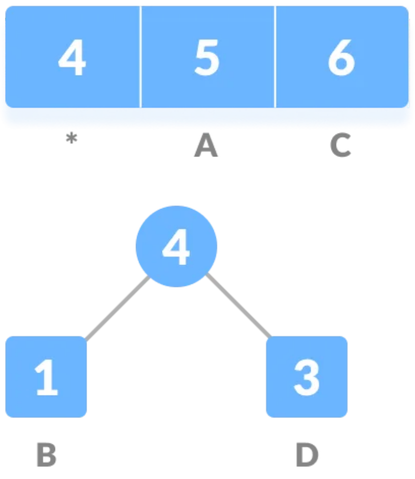
2. Sort the characters in increasing order of the frequency. These are stored in a priority queue `Q`.



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.

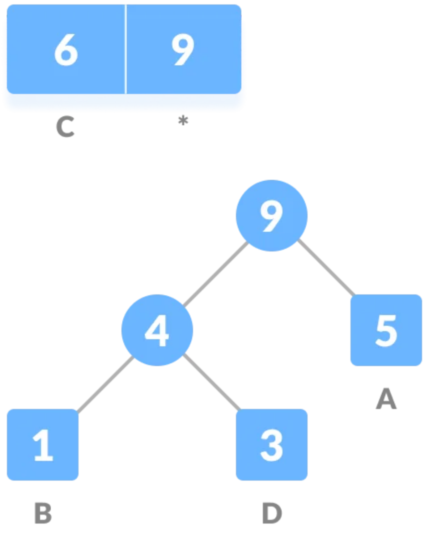


Getting the sum of the least numbers

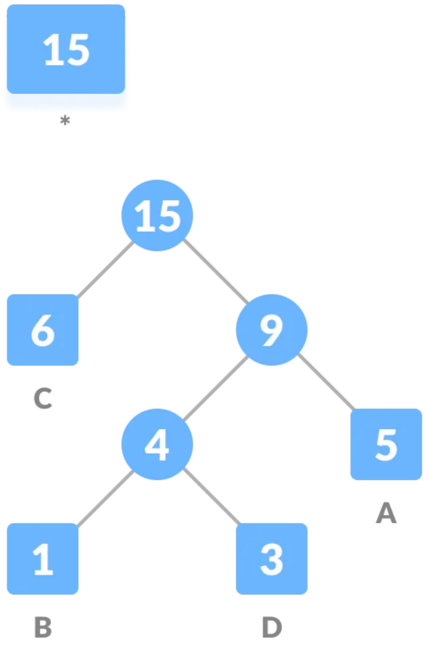
5. Remove these two minimum frequencies from `Q` and add the sum into the list of frequencies (* denote the internal nodes in the figure above).

6. Insert node `z` into the tree.

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

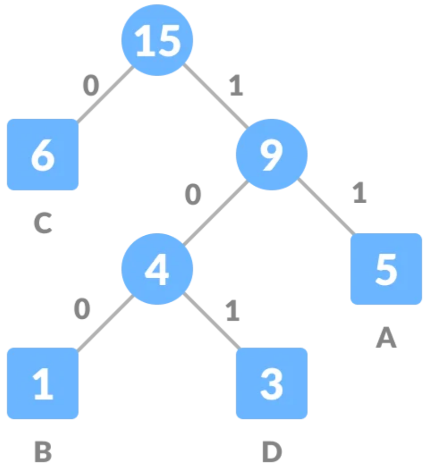


Repeat steps 3 to 5 for all the characters.



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

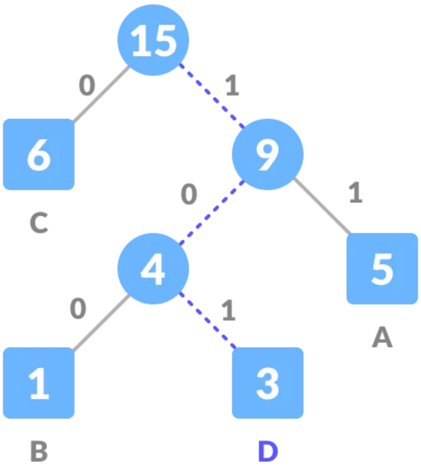
4 * 8 = 32 bits	15 bits	28 bits

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

Huffman Coding 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
```

Python, Java and C/C++ Examples

Python Java C C++

```c
// Huffman Coding in C

#include <stdio.h>
#include <stdlib.h>

#define MAX_TREE_HT 50

struct MinHNode {
  char item;
  unsigned freq;
  struct MinHNode *left, *right;
};

struct MinHeap {
  unsigned size;
  unsigned capacity;
  struct MinHNode **array;
};

// Create nodes
struct MinHNode *newNode(char item, unsigned freq) {
  struct MinHNode *temp = (struct MinHNode *)malloc(sizeof(struct MinHNode));

  temp->left = temp->right = NULL;
  temp->item = item;
  temp->freq = freq;
```

Huffman Coding Complexity

The time complexity for encoding each unique character based on its frequency is `O(nlog n)`.

Extracting minimum frequency from the priority queue takes place `2*(n-1)` times and its complexity is `O(log n)`. Thus the overall complexity is `O(nlog n)`.

Huffman Coding Applications

- Huffman coding is used in conventional compression formats like GZIP, BZIP2, PKZIP, etc.
- For text and fax transmissions.