algorithm

Huffman Coding

Huffman’s algorithm

Huffman’s algorithm is associated in creating extended binary trees that has minimum weighted path lengths from the given weights. It makes use of a table that contains frequency of occurrence for each data element.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.

The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.

- Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression.
- It compresses data very effectively saving from 20% to 90% memory, depending on the characteristics of the data being compressed.
- We consider the data to be a sequence of characters. Huffman's greedy algorithm uses a table giving how often each character occurs (i.e., its frequency) to build up an optimal way of representing each character as a binary string.
- Huffman code was proposed by David A. Huffman in 1951

There are mainly two major parts in Huffman Coding

1) Build a Huffman Tree from input characters.

2) Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

2. Extract two nodes with the minimum frequency from the min heap.

3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

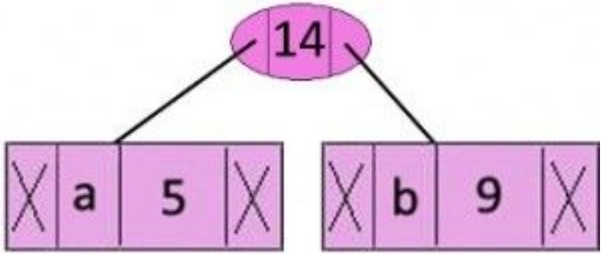
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Let us understand the algorithm with an example:

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

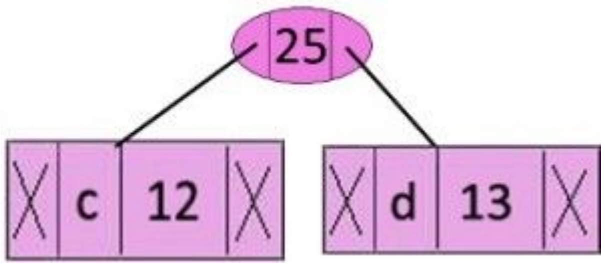
Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.



Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

character	Frequency
c	12
d	13
Internal Node	14
e	16
f	45

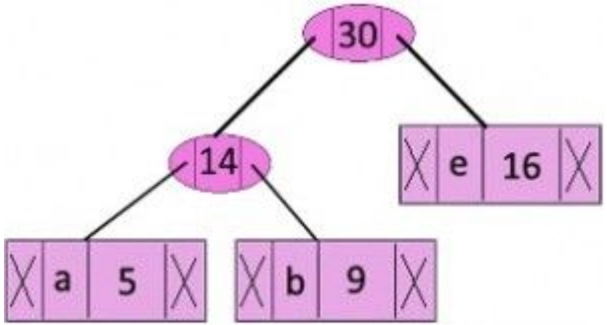
Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25



Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes.

character	Frequency
Internal Node	14
e	16
Internal Node	25
f	45

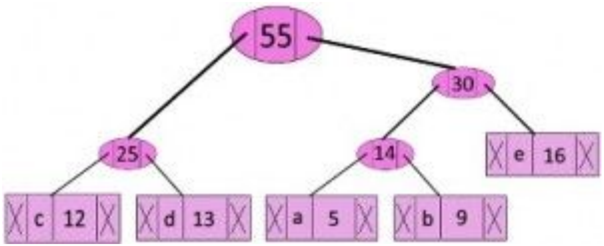
Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30



Now min heap contains 3 nodes.

character	Frequency
Internal Node	25
Internal Node	30
f	45

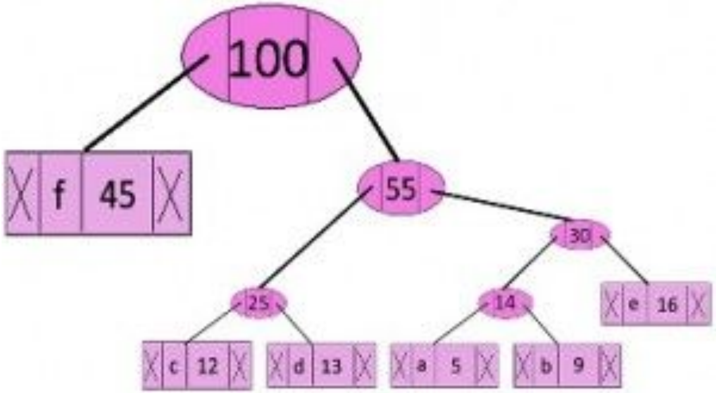
Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55



Now min heap contains 2 nodes.

character	Frequency
f	45
Internal Node	55

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100



Now min heap contains only one node.

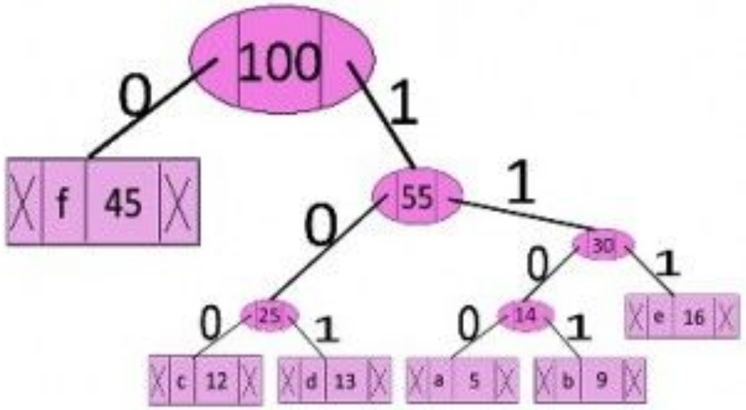
```
character       Frequency

Internal Node     100
```

Since the heap contains only one node, the algorithm stops here.

Steps to print codes from Huffman Tree:

1. Traverse the tree formed starting from the root.
2. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array.
3. Print the array when a leaf node is encountered.



The codes are as follows:

character	code-word
f	0
c	100
d	101
a	1100
b	1101
e	111

Another example: