**Huffman coding, Optimal merge patterns**

**Huffman Coding**

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

Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be “cccd” or “ccb” or “acd” or “ab”.

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

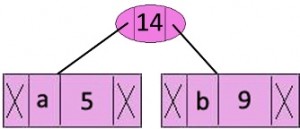
d 13

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.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/fig-1.jpeg)

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

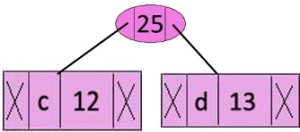
d 13

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

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/fig-2.jpg)  
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

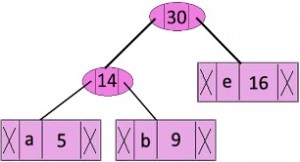
Internal Node 14

e 16

Internal Node 25

f 45

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

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/fig-3.jpg)  
Now min heap contains 3 nodes.

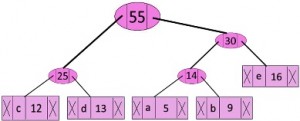
Character Frequency

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

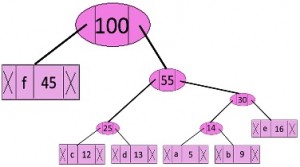
[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/fig-4.jpg)  
Now min heap contains 2 nodes.

character Frequency

f 45

Internal Node 55

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

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/fig-5.jpg)  
Now min heap contains only one node.

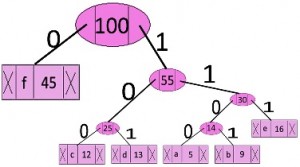
character Frequency

Internal Node 100

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

**Steps to print codes from Huffman Tree:**

Traverse the tree formed starting from the root. 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. Print the array when a leaf node is encountered.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/fig-6.jpg)  
The codes are as follows:

character code-word

f 0

c 100

d 101

a 1100

b 1101

e 111

**Optimal merge patterns**

Merge a set of sorted files of different length into a single sorted file. We need to find an optimal solution, where the resultant file will be generated in minimum time.

If the number of sorted files are given, there are many ways to merge them into a single sorted file. This merge can be performed pair wise. Hence, this type of merging is called as **2-way merge patterns**.

As, different pairings require different amounts of time, in this strategy we want to determine an optimal way of merging many files together. At each step, two shortest sequences are merged.

To merge a **p-record file** and a **q-record file** requires possibly **p + q** record moves, the obvious choice being, merge the two smallest files together at each step.

Two-way merge patterns can be represented by binary merge trees. Let us consider a set of **n** sorted files **{f1, f2, f3, …, fn}**. Initially, each element of this is considered as a single node binary tree. To find this optimal solution, the following algorithm is used.

**Algorithm: TREE (n)**

for i := 1 to n – 1 do

declare new node

node.leftchild := least (list)

node.rightchild := least (list)

node.weight) := ((node.leftchild).weight) + ((node.rightchild).weight)

insert (list, node);

return least (list);

At the end of this algorithm, the weight of the root node represents the optimal cost.

## **Example**

Let us consider the given files, f1, f2, f3, f4 and f5 with 20, 30, 10, 5 and 30 number of elements respectively.

If merge operations are performed according to the provided sequence, then

**M1 = merge f1 and f2** => 20 + 30 = 50

**M2 = merge M1 and f3** => 50 + 10 = 60

**M3 = merge M2 and f4** => 60 + 5 = 65

**M4 = merge M3 and f5** => 65 + 30 = 95

Hence, the total number of operations is

50 + 60 + 65 + 95 = 270

Now, the question arises is there any better solution?

Sorting the numbers according to their size in an ascending order, we get the following sequence −

**f4, f3, f1, f2, f5**

Hence, merge operations can be performed on this sequence

**M1 = merge f4 and f3** => 5 + 10 = 15

**M2 = merge M1 and f1** => 15 + 20 = 35

**M3 = merge M2 and f2** => 35 + 30 = 65

**M4 = merge M3 and f5** => 65 + 30 = 95

Therefore, the total number of operations is

15 + 35 + 65 + 95 = 210

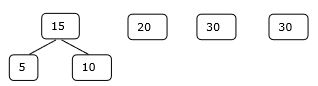
Obviously, this is better than the previous one.

In this context, we are now going to solve the problem using this algorithm.

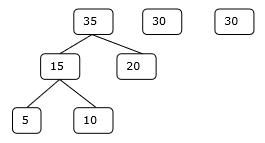
### **Initial Set**

Initial Set

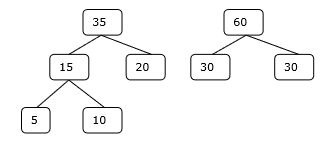
### **Step-1**



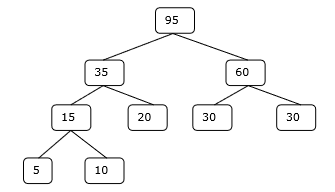
### **Step-2**



### **Step-3**



### **Step-4**



Hence, the solution takes 15 + 35 + 60 + 95 = 205 number of comparisons.

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. <https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>
2. <https://www.programiz.com/dsa/huffman-coding>
3. <https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_optimal_merge_pattern.htm#:~:text=Merge%20a%20set%20of%20sorted,into%20a%20single%20sorted%20file.>

**Lecture Video:**

1. <https://youtu.be/0kNXhFIEd_w>
2. <https://youtu.be/HHIc5JZyenI>

**Online Notes:**

1. <http://vssut.ac.in/lecture_notes/lecture1428551222.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.

**In addition: PPT can be also be given.**