AssignmentNo.	**2**
Title	To Implement Huffman Encoding using a greedy strategy.
PROBLEM STATEMENT/DEFINITION	Write a program to implement Huffman Encoding using a greedy strategy.
Objectives	Understand and implement the concept of Huffman encoding
Software packages and hardware apparatus used	Technology : Java/Python Ubuntu /Linux Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
References	1. http://en.wikipedia.org/wiki/Huffman_coding 2. https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/#:~:text=Steps%20to%20build%20Huffman%20Tree&text=Create%20a%20new%20internal%20node,heap%20contains%20only%20one%20node.
STEPS	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.

	Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept, Architecture, Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

Concepts Related Theory

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 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.

4. 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)
5. Extract two nodes with the minimum frequency from the min heap.

6. 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.
7. 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:

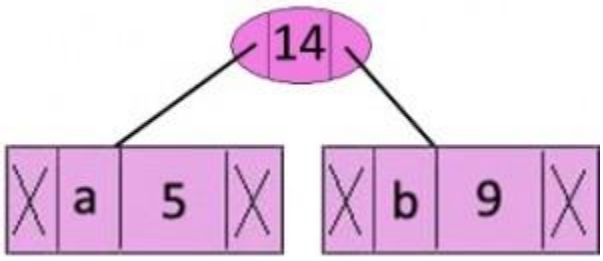
character	Frequency
a	5
b	9
c	12
d	13

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.

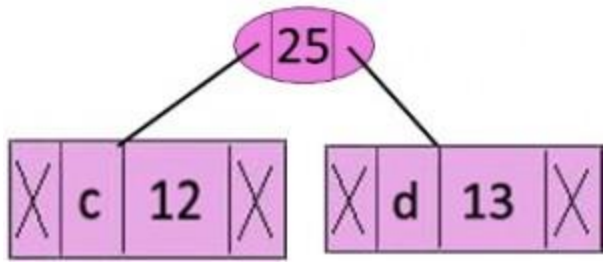
Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.



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

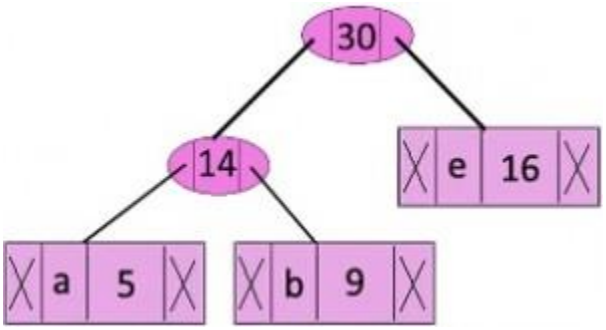
Internal Node 14

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.

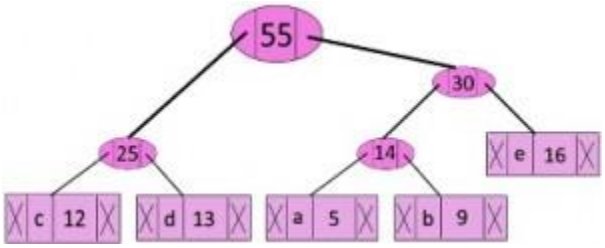
character Frequency

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



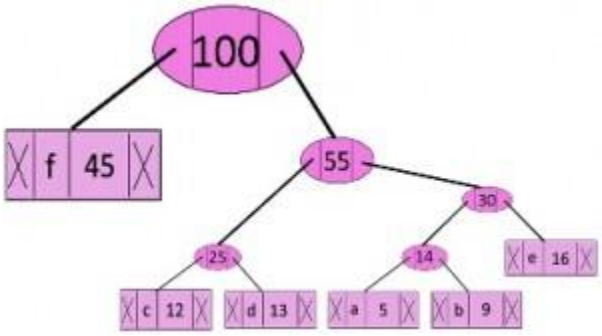
Now min heap contains 2 nodes.

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.

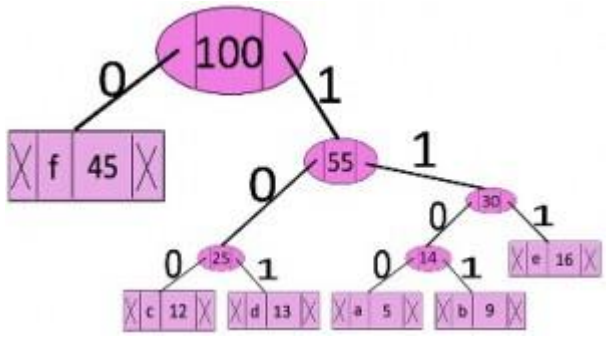
character Frequency

Internal Node 100

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

Steps to print codes from Huffman Tree:

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.



The codes are as follows:

character code-word

f	0
c	100
d	101
a	1100
b	1101
e	111

Pseudocode

```
# A Huffman Tree Node
class node:
        def __init__(self, freq, symbol, left=None, right=None):
```

```
        # frequency of symbol
        self.freq = freq

        # symbol name (character)
        self.symbol = symbol

        # node left of current node
        self.left = left

        # node right of current node
        self.right = right

        # tree direction (0/1)
        self.huff = ''

# utility function to print huffman
# codes for all symbols in the newly
# created Huffman tree


def printNodes(node, val=''):
    # huffman code for current node
    newVal = val + str(node.huff)

    # if node is not an edge node
    # then traverse inside it
    if(node.left):
        printNodes(node.left, newVal)
    if(node.right):
        printNodes(node.right, newVal)
```

```
            # if node is edge node then
            # display its huffman code
        if(not node.left and not node.right):
            print(f"{node.symbol} -> {newVal}")


# characters for huffman tree
chars = ['a', 'b', 'c', 'd', 'e', 'f']

# frequency of characters
freq = [ 5, 9, 12, 13, 16, 45]

# list containing unused nodes
nodes = []

# converting characters and frequencies
# into huffman tree nodes
for x in range(len(chars)):
        nodes.append(node(freq[x], chars[x]))

while len(nodes) > 1:
        # sort all the nodes in ascending order
        # based on their frequency
        nodes = sorted(nodes, key=lambda x: x.freq)

        # pick 2 smallest nodes
        left = nodes[0]
        right = nodes[1]

        # assign directional value to these nodes
        left.huff = 0
```

```
        right.huff = 1

        # combine the 2 smallest nodes to create
        # new node as their parent
        newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

        # remove the 2 nodes and add their
        # parent as new node among others
        nodes.remove(left)
        nodes.remove(right)
        nodes.append(newNode)

# Huffman Tree is ready!
printNodes(nodes[0])
```

Output:

f: 0

c: 100

d: 101

a: 1100

b: 1101

e: 111

Time complexity*:* O(nlogn) where n is the number of unique characters. If there are n nodes, extractMin() is called 2*(n – 1) times. extractMin() takes O(logn) time as it calles minHeapify(). So, overall complexity is O(nlogn).

If the input array is sorted, there exists a linear time algorithm. We will soon be discussing in our next post.

Applications of Huffman Coding:

1. They are used for transmitting fax and text.
2. They are used by conventional compression formats like PKZIP, GZIP, etc.
3. Multimedia codecs like JPEG, PNG, and MP3 use Huffman encoding(to be more precise the prefix codes).

It is useful in cases where there is a series of frequently occurring characters.

FAQ's

Q-1.Which of the following is true about Huffman Coding?

(A)Huffman coding may become lossy in some cases

(B)Huffman Codes may not be optimal lossless codes in some cases

(C)In Huffman coding, no code is prefix of any other code.

(D) All of the above

Q-2. How many bits may be required for encoding the message 'mississippi'?

Q-3 How can you generate Huffman codes using greedy method?

Q-4 What is the greedy choice in Huffman coding?

Q-5 Is Huffman algorithm A greedy algorithm?

Q-6 Is Huffman coding lossy or lossless?

Q-7 Which technique is applied to compress the given message using greedy?

Q-8 What is the running time of Huffman encoding algorithm?

Q-9 What are the various applications of Huffman coding?

Q-10 What is the advantage of Huffman code over variable length code?