Experiment No.:-2

Write a program to implement Huffman Encoding using a greedy strategy.

Source Code:-

In [1]:

```
import heapq

class Node:     def __init__(self, freq, symbol, left=None,
right=None):
        self.freq = freq
self.symbol = symbol
self.left = left
self.right = right
self.huff = ""
        def __lt__(self,
other):
        return self.freq < other.freq

def printNodes(node, val=""):
newval = val + node.huff     if
node.left or node.right:
if node.left:
            printNodes(node.left, newval)
if node.right:
            printNodes(node.right, newval)
else:
        print(f"{node.symbol} -> {newval}")
encoded_lengths[node.symbol] = len(newval)

# Getting user input for characters and their frequencies
num_chars = int(input("Enter number of characters: "))
chars = [] freqs = []

for i in range(num_chars):
    char = input(f"Enter character {i + 1}: ")
    freq = int(input(f"Enter frequency of character {char}: "))
chars.append(char)     freqs.append(freq) nodes = []

for i in range(len(chars)):
heapq.heappush(nodes, Node(freqs[i], chars[i]))

while len(nodes) > 1:
    left = heapq.heappop(nodes)
right = heapq.heappop(nodes)
```

```
    left.huff = "0"     right.huff = "1"     newnode = Node(left.freq + right.freq,
left.symbol + right.symbol, left, right)     heapq.heappush(nodes, newnode)

# Calculating total size before encoding total_size_before
= sum(freqs) * 8

# Printing the nodes and calculating encoded lengths
encoded_lengths = {} printNodes(nodes[0])

# Calculating total size after encoding
total_size_after = sum(freqs[i] * encoded_lengths[chars[i]] for i in range(num_chars))

# Calculating Encoded Data Representation
characters = num_chars * 8 frequency =
sum(freqs)
encoded_data_representation = characters + frequency + total_size_after

print("\nTotal size before encoding:", total_size_before, "bits")
print("Total size after encoding:", total_size_after, "bits") print("Encoded
Data Representation:", encoded_data_representation, "bits")
```

```
Enter number of characters: 4
Enter character 1: B
Enter frequency of character B: 1
Enter character 2: C
Enter frequency of character C: 6
Enter character 3: A
Enter frequency of character A: 5 Enter
character 4: D
Enter frequency of character D: 3
C -> 0 B
-> 100 D
-> 101
A -> 11

Total size before encoding: 120 bits
Total size after encoding: 28 bits
Encoded Data Representation: 75 bits
```

In [ ]: