2.Write a program to implement Huffman Encoding using a greedy strategyImplementation

```
def printNodes(node, val=''):
        newVal = val + str(node.huff)
        if(node.left):
                printNodes(node.left, newVal)
        if(node.right):
                printNodes(node.right, newVal)
        if(not node.left and not node.right):
                print(f"{node.symbol} -> {newVal}")
```

- # characters for huffman tree

```
chars = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
# frequency of characters
freq = [ 4, 7, 12, 14, 17, 43, 54]
# 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

a -> 0000 b -> 0001 c -> 001

d -> 010

e -> 011

f -> 10

g -> 11