

## DAA HUFFMAN

```
import heapq

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 = ""

    def __lt__(self, nxt):

        return self.freq < nxt.freq


# 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)):
```

```
    heapq.heappush(nodes, node(freq[x], chars[x]))
```

```
while len(nodes) > 1:

    # sort all the nodes in ascending order
    # based on their frequency
    left = heapq.heappop(nodes)
    right = heapq.heappop(nodes)

    # 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)

    heapq.heappush(nodes, newNode)

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