```python
import heapq

class Node:
    def __init__(self, freq, symbol, left=None, right=None):
        self.freq = freq            # Frequency of symbol
        self.symbol = symbol        # Symbol name (character)
        self.left = left            # Node left of current node
        self.right = right          # Node right of current node
        self.huff = ''              # Tree direction (0/1)

    def __lt__(self, nxt):          # Check if current frequency is 
less than the next node's
        return self.freq < nxt.freq

def print_nodes(node, val=''):
    newval = val + str(node.huff)

    # If node is not a leaf node, traverse it
    if node.left:
        print_nodes(node.left, newval)
    if node.right:
        print_nodes(node.right, newval)

    # If node is a leaf node, display its Huffman code
    if not node.left and not node.right:
        print(f"{node.symbol} -> {newval}")

if __name__ == "__main__":
    chars = ['a', 'b', 'c', 'd', 'e', 'f']
    freq = [5, 9, 12, 13, 16, 45]

    nodes = []

    # Converting characters and frequencies into heap nodes
    for i in range(len(chars)):
        heapq.heappush(nodes, Node(freq[i], chars[i]))

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

        left.huff = 0
        right.huff = 1

        # Combine the 2 smallest nodes to create a new parent node
        newnode = Node(left.freq + right.freq, left.symbol + 
right.symbol, left, right)

        heapq.heappush(nodes, newnode)
    # Print the Huffman codes
    print_nodes(nodes[0])  # Passing root of Huffman Tree
```

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