```
# Huffman Encoding (Recursion, No Tree)

def huffman(prob, grp, M, code):

    if len(prob) == 1:
        return code


    pick_p = prob[-M:]
    pick_g = grp[-M:]


    for d in range(M):

        digit = M - 1 - d
        sym = pick_g[-(d + 1)]

        for s in sym:
            code[s] = str(digit) + code[s]


    new_p = sum(pick_p)
    new_g = []

    for g in pick_g:
        new_g.extend(g)
```

```
    prob = prob[:-M]
    grp = grp[:-M]


    prob.append(new_p)
    grp.append(new_g)


    temp = list(zip(prob, grp))
    temp.sort(reverse=True, key=lambda x: x[0])

    prob, grp = zip(*temp)

    prob = list(prob)
    grp = list(grp)


    return huffman(prob, grp, M, code)



# -------- MAIN --------

prob = [0.30, 0.25, 0.20, 0.15, 0.10]
M = 2

N = len(prob)
```

```
grp = [[i + 1] for i in range(N)]


code = [""] * (N + 1)



while (N - 1) % (M - 1) != 0:

    prob.append(0)

    grp.append([])

    N += 1



temp = list(zip(prob, grp))

temp.sort(reverse=True, key=lambda x: x[0])


prob, grp = zip(*temp)


prob = list(prob)

grp = list(grp)



code = huffman(prob, grp, M, code)
```

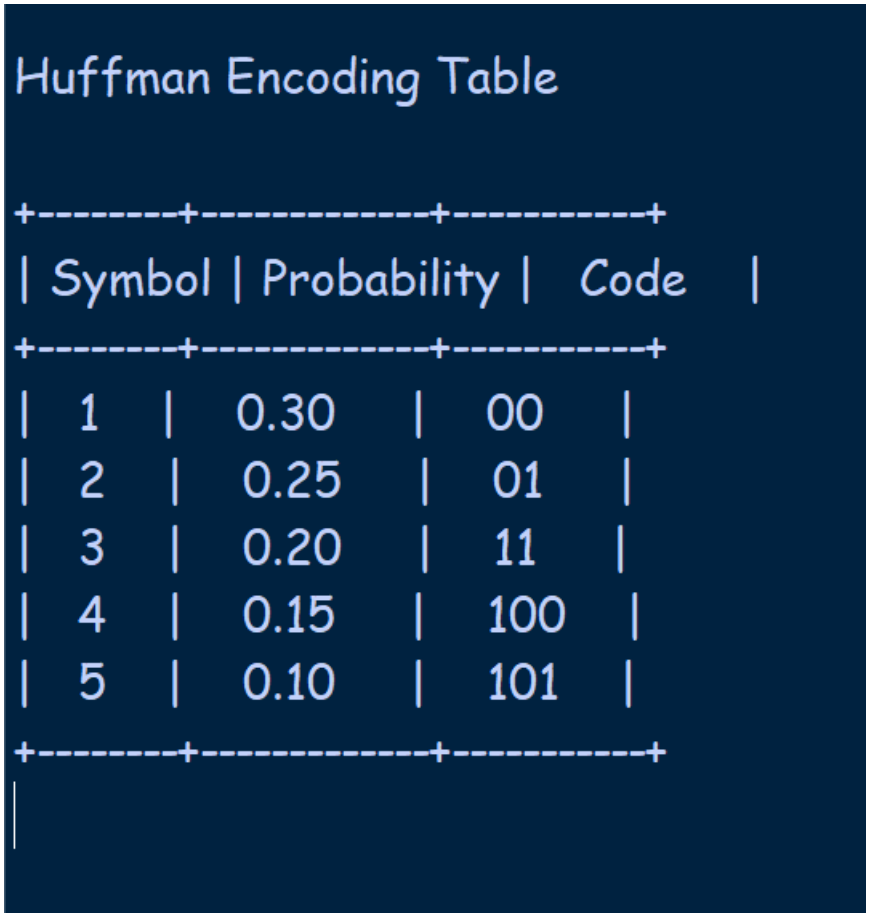
```
# -------- OUTPUT TABLE --------

print("\nHuffman Encoding Table\n")

print("+--------+-------------+-----------+")
print("| Symbol | Probability |   Code    |")
print("+--------+-------------+-----------+")

for i in range(1, len(code)):
    print(f"| {i:^6} | {prob[i-1]:^11.2f} | {code[i]:^9} |")
```

print("+--------+-------------+-----------+")


Huffman Encoding Table
+--------+-------------+-----------+
| Symbol | Probability | Code |
+--------+-------------+-----------+
1	0.30	00
2	0.25	01
3	0.20	11
4	0.15	100
5	0.10	101
+--------+-------------+-----------+