Huffman

This one I used chatgpt 100% because I'm noob at cryptography. ChatGPT is a tool tho so don't shy shy to use it. But you need to know what you are looking for. You need to guide the ChatGPT.

```
from heapq import heappush, heappop

# Example probabilities dictionary
probabilities = {'co': 0.007352941176470588, 'me': 0.007352941176470588, 'e ': 0.01838235294117647, ' t':
0.01838235294117647, 'to': 0.011029411764705883, 'o ': 0.011029411764705883, 'ou': 0.01838235294117647, 'ut':
0.007352941176470588, 's ': 0.025735294117647058, 'CT': 0.007352941176470588, 'TF': 0.007352941176470588, ' I':
0.011029411764705883, "I'": 0.007352941176470588, "'m": 0.007352941176470588, 'm ': 0.011029411764705883, ' s':
0.007352941176470588, ' h': 0.007352941176470588, 'ha': 0.007352941176470588, 'y ': 0.007352941176470588, ' y':
0.007352941176470588, 'yo': 0.007352941176470588, ' w': 0.007352941176470588, 'er': 0.007352941176470588, 're':
0.011029411764705883, ' a': 0.014705882352941176, 'ab': 0.007352941176470588, 'le': 0.007352941176470588, 't ':
0.022058823529411766, 'th': 0.014705882352941176, 'hi': 0.007352941176470588, 'is': 0.011029411764705883, ' m':
0.007352941176470588, 'es': 0.007352941176470588, 'ss': 0.007352941176470588, 'ag': 0.007352941176470588, 'e.':
0.007352941176470588, '. ': 0.011029411764705883, 'as': 0.011029411764705883, ' i': 0.014705882352941176, 'it':
0.011029411764705883, 'ar': 0.007352941176470588, 'ur': 0.007352941176470588, 'ne': 0.007352941176470588, 'd ':
0.007352941176470588, ' o': 0.007352941176470588, 'on': 0.007352941176470588, ' c': 0.007352941176470588, 'la':
0.007352941176470588, 'wa': 0.007352941176470588, '..': 0.007352941176470588, 'W': 0.022058823529411766, 'e':
0.058823529411764705, 'l': 0.025735294117647058, 'c': 0.01838235294117647, 'o': 0.058823529411764705, 'm':
0.022058823529411766, ' ': 0.13970588235294118, 't': 0.05514705882352941, 'U': 0.007352941176470588, '-':
0.003676470588235294, 'S': 0.007352941176470588, 'u': 0.022058823529411766, '"': 0.011029411764705883, 's':
0.05514705882352941, 'C': 0.011029411764705883, 'T': 0.01838235294117647, 'F': 0.007352941176470588, '!':
0.007352941176470588, 'I': 0.011029411764705883, 'h': 0.025735294117647058, 'a': 0.051470588235294115, 'p':
0.014705882352941176, 'y': 0.029411764705882353, 'w': 0.011029411764705883, 'r': 0.03308823529411765, 'b':
0.014705882352941176, 'd': 0.014705882352941176, 'i': 0.025735294117647058, 'g': 0.01838235294117647, ':':
0.022058823529411766, '?': 0.003676470588235294, 'n': 0.025735294117647058, 'f': 0.007352941176470588, 'A':
0.007352941176470588, 'H': 0.003676470588235294, ':': 0.003676470588235294, 'O': 0.003676470588235294, '{':
0.003676470588235294, '0': 0.007352941176470588, 'L': 0.003676470588235294, 'Z': 0.003676470588235294, 'v':
0.003676470588235294, 'E': 0.003676470588235294, '2': 0.003676470588235294, '3': 0.003676470588235294, 'N':
0.007352941176470588, 'K': 0.003676470588235294, 'k': 0.003676470588235294, '8': 0.007352941176470588, 'J':
0.007352941176470588, '6': 0.007352941176470588, 'M': 0.003676470588235294, 'x': 0.003676470588235294, '7':
0.003676470588235294, '}': 0.003676470588235294}

# Actual Huffman encoded message
encoded_message =
"01110111101100111110011101100101001100001101000001001011101111110100111101110111101000111110000101010001
01100101101011100000101011010111011111111111010110100000011000011000011000001111011111101100111001001101
10110000100101101001111001101011000011010011011011110101011001000111100000110100011110111001010111100000
00000000110110111111111111111100001011101011111111100001000110000110010001101101111101110110101011101111
00110100111000100011101101110011111111100011010100111001101000111010111110000001111010011010011100011
10111001011111111010100111010100110000100001111101011101000011100110101001110100101011111011010010111
10010000000000011101011000010100000100000000111110101010010000011110111001001101010010111010001101111
01100100101111111011110011000110001001111100111110000011010101101001111100000001011110010110011011011
01000011011000010010010111111111100001001101000001111110010001010010100000100001111011101010100010001
01001010101110010110101110110010001110101010001011110011000110011001010101101111010111001101011101111
11011111001100110101000101101110001111010111110101110100000111001011110010011110001110111100100111001011
01110100010111111000111010111101011011111101101100000100011011010111110101001101011001000101110100010
10100010001001010101101100000101"


class Node:
    def __init__(self, symbol, freq):
        self.symbol = symbol
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):  # Corrected method name
        return self.freq < other.freq

def build_huffman_tree(probabilities):
    pq = []
    for symbol, freq in probabilities.items():
```

```
        heappush(pq, Node(symbol, freq))

    if len(pq) == 1:
        return pq[0]  # Handle the case with a single type of symbol

    while len(pq) > 1:
        left = heappop(pq)
        right = heappop(pq)
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heappush(pq, merged)

    return pq[0] if pq else None

def generate_codes(node, prefix="", code_map={}):
    if node is not None:
        if node.symbol is not None:
            code_map[node.symbol] = prefix
        generate_codes(node.left, prefix + "0", code_map)
        generate_codes(node.right, prefix + "1", code_map)
    return code_map

def decode_huffman(encoded, code_map):
    reverse_map = {v: k for k, v in code_map.items()}
    current_code = ""
    decoded_message = ""

    for bit in encoded:
        current_code += bit
        if current_code in reverse_map:
            decoded_message += reverse_map[current_code]
            current_code = ""  # Reset the current code after decoding
    return decoded_message


# Running the Huffman Tree functions
root = build_huffman_tree(probabilities)
if root:
    code_map = generate_codes(root)
    decoded_message = decode_huffman(encoded_message, code_map)
    print("Decoded Message:", decoded_message)  # Print the decoded message
else:
    print("Failed to build Huffman tree.")
```

```
$ python3 script.py
Decoded Message: Welcome to UW-Stout's CTF! I'm so happy you were able to decrypt this message. Was it hard? I'm not sure. I learned about this algorithm in one of my classes and thought it was cool...Anyways. Here is your flag:STOUTCTF{A0LZTvEW23NcbeKk8JyWJ8W0b6Mx7p6N}Congrats!
```

You learned in your classes? Damn Im jealous

STOUTCTF{A0LZTvEW23NcbeKk8JyWJ8W0b6Mx7p6N}