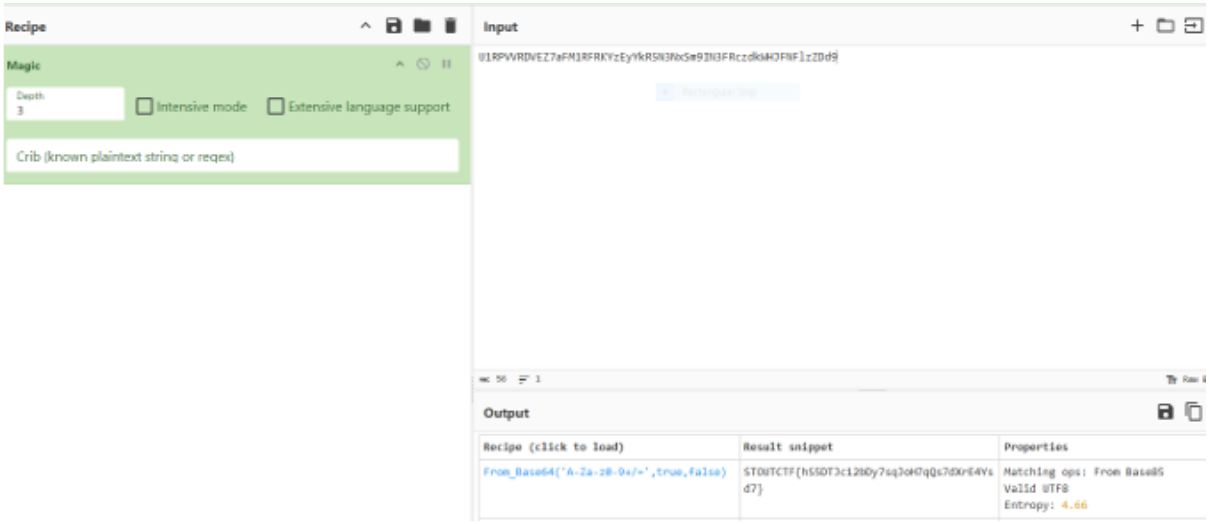
Question: Based

copy the text file to cyberchef and put magic recipe and you got the flag



Flag: STOUTCTF{hS5DTJc12bDy7sqJoH7qQs7dXrE4Ysd7}

Question: V

v.txt
Use the Giovan alphabet provided to you ABCDEFGHIJKLMNOPQRSTUVWXYZ

Can you solve the great GIOVAN mystery?

YBCPTPZN{ EaT8CK2zVexEqjdCmP6URd14xW6kNg7B}

So I was searching for Giovan on chatgpt and it told me about vigenere cipher so I tried it using dcode

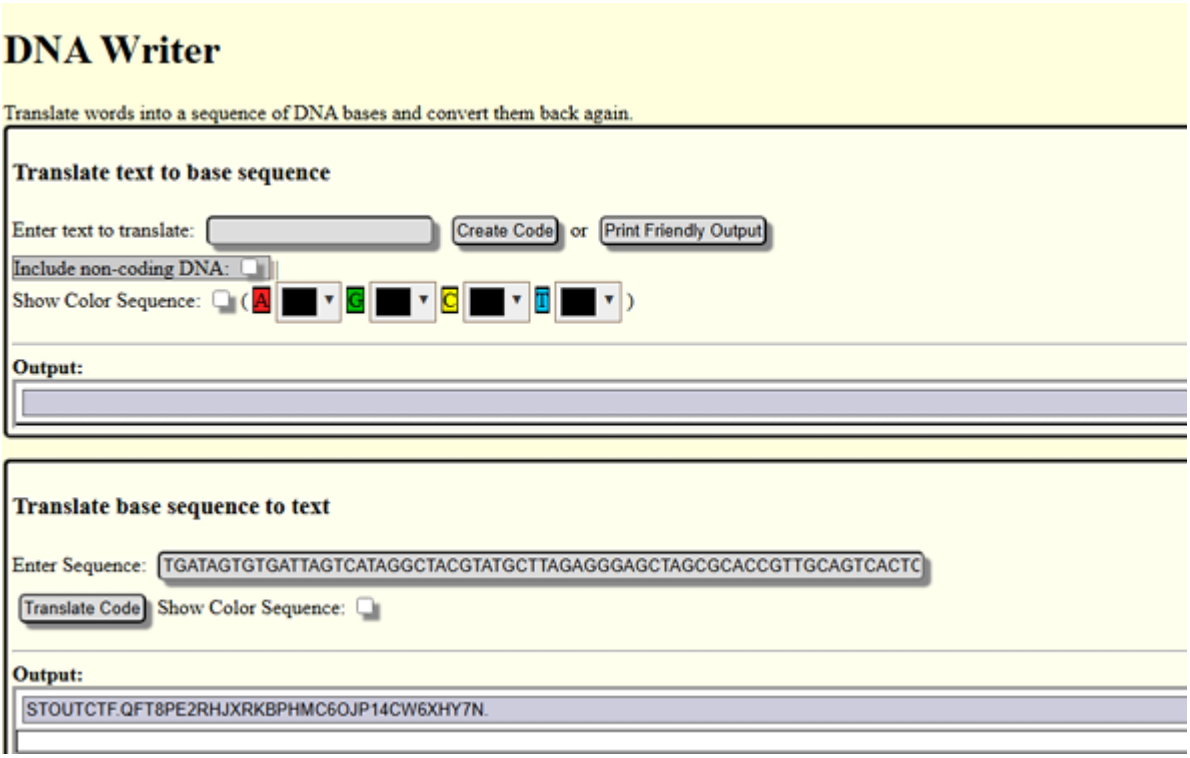
Flag: STOUTCTF{QfT8PE2rHjxRkbpHmC6OJp14cW6xHy7N}

Question: Jeans

Jeans.txt

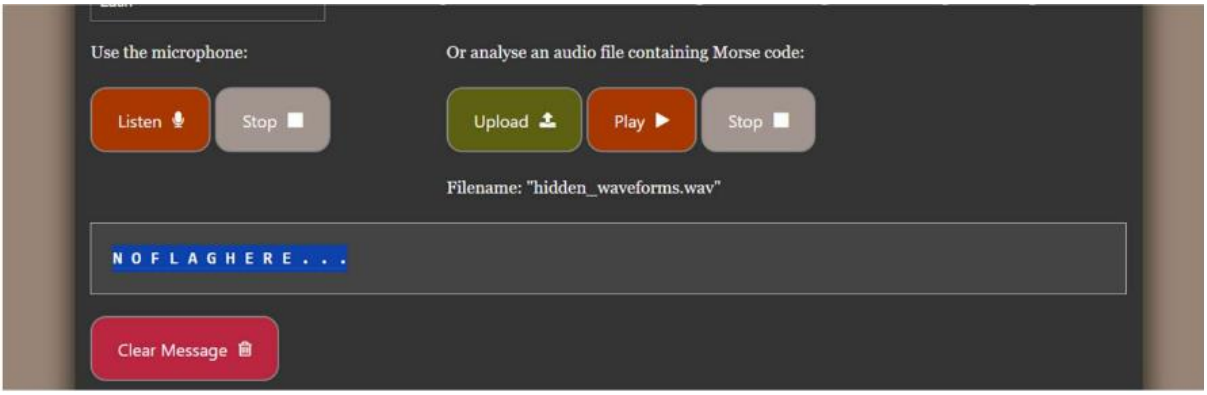
TGATAGTGTGATTAGTCATAGGCTACGTATGCTTAGAGGGAGCTAGCGCACCGTTGCAGTCAC
TCGCATGAGCGTACATCAATTTGTTGCGAGTCTAGATCAATGATTAGTCGTGACACCCTCACG

Jeans sounds like gene ?? so I went to find a DNA decoder online and tried it and got the flag

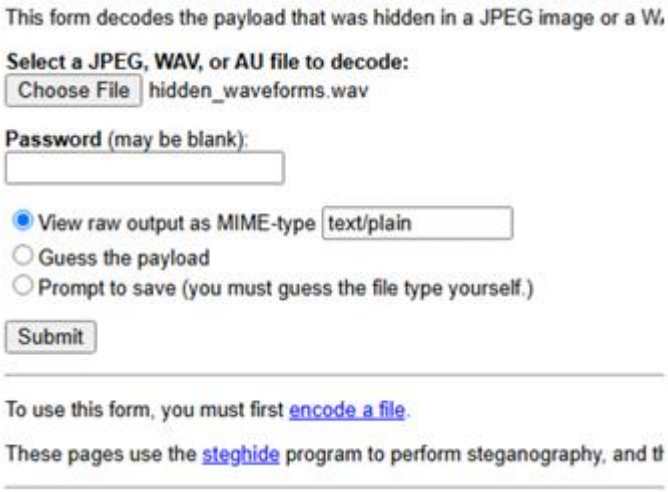


Flag: STOUTCTF{QFT8PE2RHJXRKBPHMC6OJP14CW6XHY7N}

Question: Hidden Waveforms



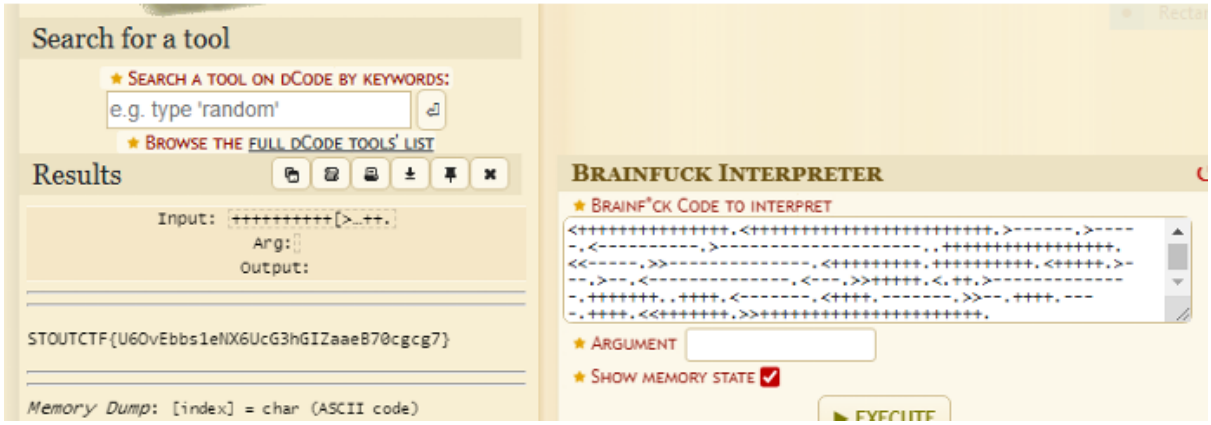
First I hear the sound first and looks like morse code. But what I got is NOFLAGHERE...



Then I try use stega tools online to extract any hidden strings or anything

This is what I got

After that I go decode this decimal



it was brainfuck cipher.and I used dcode to copy paste and decode it.

Flag: STOUTCTF{U6OvEbbs1eNX6UcG3hGIZaaeB70cgcg7}

Question: Custom Cipher

Your scripting cant break my encoding!

VWRXWFWI{Vr3J8NJks4Tn58PjDv3IPNc9VueGFwlu}

Go to Caesar Cipher in Dcode and paste the encoded flag and you'll get the flag

Flag= STOUTCTF{So0G5KGhp1Qk25MgAs0FMK96SrbDCtir}

Question: Fat Finger

DYPIYVYG}fyYV0[s-KhuDwV[OKn:Y<H:4k8CsYeFY|

I googled what is fat finger

A "fat-finger" error, or "fat-fingering," is a slang term for a typing mistake caused by accidentally pressing the wrong key on a keyboard or touchscreen, often due to keys being too close together

so I rearrange it to get the flag

flag= STOUTCTF{dtTC9pa0JgySqCpIJbLTMGL3j7XaTwDT}

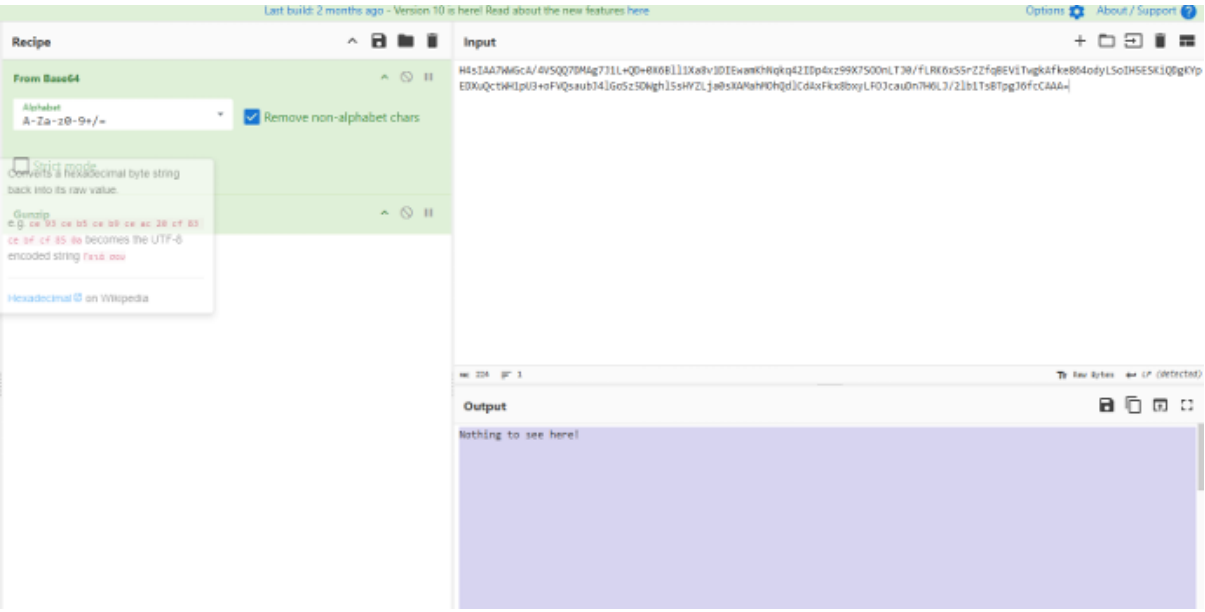
Question: Nothing To See Here !

I have been given an empty file but when I can hover over it so after researching for awhile it was a whitespace language

I went to dcode and paste it

I got a based64
H4sIAA7WWGcA/4VSQQ7DMAg7J1L+QD+0X6Bll1Xa8v1DIEwamKhNqkq42IDp4xz99X7SOOnLTJ0/fLRK6xS5rZZfqBEViTwgkAfke864odyLSoIH5ESKiQDgKYpEDXuQctWH1pU3+oFVQsaubJ4lGoSz5DWghl5sHYZLja0sXAMahMOhQdlCdAxFkx8bxyLFOJcauDn7H6LJ/2lb1TsBTpgJ6fcCAAA=

so I went to cyberchef to decode it



Oops it says that Nothing to see here but oh well I tried to hover it again and I cld hover over it again so I used the white space language on dcode again.

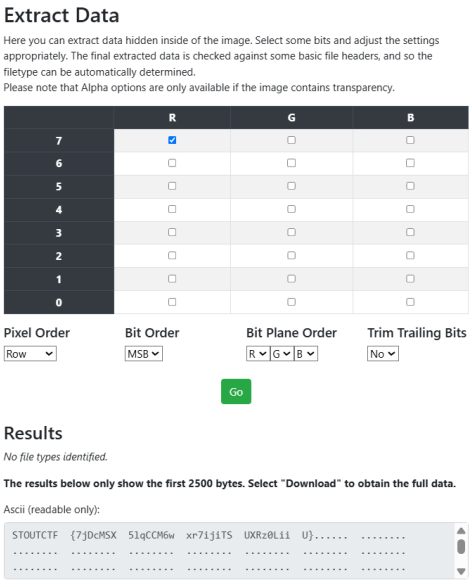
And I got the flag !

flag= STOUTCTF{ab6IcT8R4vNyAJNWQteBJ3Yd2VTrkVCp}

Question: 7 Bit Flow

We got a picture

For this challenge all u need to do is extract the data using red and choose 7 in this website (https://georgeom.net/StegOnline/extract) and you'll get the flag

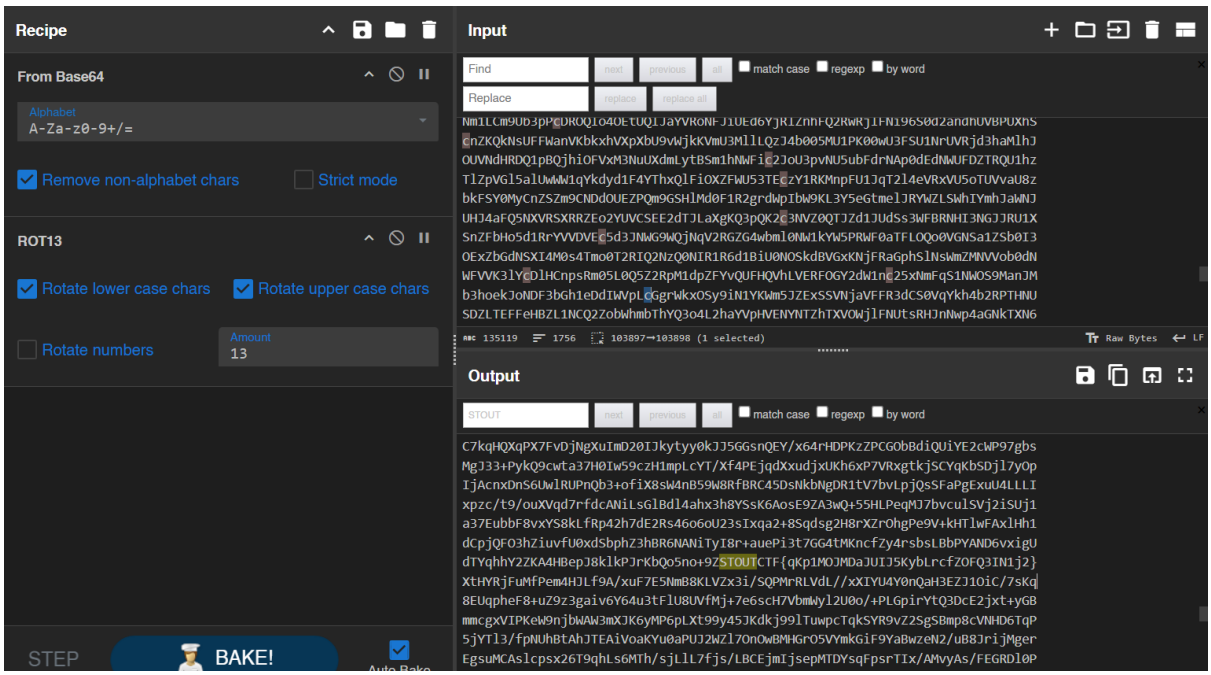


Flag= STOUTCTF {7jDcMSX5lqCCM6wxr7ijiTSUXRz0LiiU}

Question: 13RottenTermites

I got a looooong text file so I went to cyberchef

I went from base64 to decrypt to rot13 because it's the hint from the challenge name and I found the Flag hidden in the long text files.



Flag= STOUTCTF{qKp1MOJMDaJUIJ5KybLrcfZOFQ3IN1j2}

Question: Huffman

I have been given a weird encoded file so I googled what Huffman is and after some researching i found a Huffman script that I could use

solve.py

```
from heapq import heappush, heappop

class Node:
    def __init__(self, symbol, freq):  # Corrected constructor method name
        self.symbol = symbol
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        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)
        print(f"Merged nodes: ({left.symbol}, {left.freq}) + ({right.symbol}, {right.freq})")

    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

# 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 =
"011101111011001111100111011001010011000011010000010010111011111101001111011101111010001111100001010100
011001011010111000001010110101110111111111110101101000000110000110000110000011110111111011001110010011
101100001001011010011110011010110000110100110110111101010110010001111000001101000111101110010111100000
000000001101101111111111111110000010111010111111111000010001100001100100011011011111101110110101110111111
001101001110001000111011011100111111111000110101001110011010001110101111100000011110100110100111000111
101110010111111110101001110101001100001000011111010111010000111001101010011101001010111110110100101111
100100000000000111010110000101000001000000001111101010100100000111101110010011010100101110100011011111
011001001011111110111100110001100010011111001111100000110101011010011111000000010111100101100110110110
010000110110000100100101111111111000010011010000011111100100101001010000010000111011101010100010001000
010010101011100101101011101100100011101010100010111100110001100110010101011011110101110011010111011011
110111110011001101010001011011100011110111111010111010000011100101111001001111000111011110010011100101
011101000101111100011101011111010110111111011011010000100011011010111110101001101011001000101110100010
101000100010010101011011000001 01"

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

Output of the script

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!

Flag: STOUTCTF{A0LZTvEW23NcbeKk8JyWJ8W0b6Mx7p6N}