Homework 6: Group Programming Assignment

Instruction:

- This is a group assignment. You must form a group of 2 or 3 people.
- The assignment must be written in Python, C, C++, or Java.
- Submit your code file (not a PDF) via Canvas.
- Only one student from the group needs to submit the assignment.
- The first line of your code should contain the full names and UCIDs of all team members, in a comment.
- Submissions must be made through Canvas before the due date. Do not email your submission.
- Any submission with a plagiarism score over 90% will receive a grade of zero, without negotiation or exception.

Please be aware: I will NOT accept submissions via email and will not open them. Submit your work on time and through Canvas only.
Using ChatGPT or any AI tool will result in a grade of ZERO without negotiation or exception.

You are given a text of at most 256 characters. Implement the following functions as part of this assignment:

a) `frequency_table(st)`

Write a function `frequency_table(st)` that constructs a frequency table for each character in the string `st` and prints it.
Input: A string `st` with a maximum length of 256 characters.
Output: Print each character along with its corresponding frequency.

b) `Huffman_code(st)`

Write a function `Huffman_code(st)` that builds a Huffman tree from the string `st` and generates the Huffman codes for each character. Print each character with its corresponding code.
Input: A string `st` with a maximum length of 256 characters.
Output: Print the Huffman code for each character in the string.

c) `Huffman_encode(st,codes)`

Write a function `Huffman_encode(st,codes)` that prints the binary encoded version of `st`, based on the Huffman codes generated in part (b) `codes`.
Input: A string `st` with a maximum length of 256 characters and its Huffman codes `codes`.
Output: Print the binary-encoded string using the Huffman codes.

d) `Huffman_tree(L)`

You are given a list `L` of characters and their corresponding Huffman codes. Write a function `Huffman_tree(L)` that constructs a Huffman tree based on this list.
Input: A list `L` of characters and their corresponding Huffman codes.
Output: Return the Huffman tree constructed from the list.

e) `Huffman_decode(bst,tree)`

Using the Huffman tree built in part (d), write a function `Huffman_decode(bst,tree)` to decode the binary-encoded text `bst` back into its original string.
Input: A binary-encoded string `bst` and its Huffman tree `tree`.
Output: Print the decoded string.

Example:

```
st = "abbcccdddd"

# a) frequency_table(st)
# Input:
frequency_table(st)

# Output:
# Character Frequencies:
# 'a': 1
# 'b': 2
# 'c': 3
# 'd': 4
# b) Huffman_code(st)
# Input:
Huffman_code(st)

# Output:
# Huffman Codes:
# 'a': 000
# 'b': 001
# 'c': 01
# 'd': 1


# c) Huffman_encode(st)
# Input:
Huffman_encode(st, codes)

# Output:
# Encoded String:
# 000001001001010101111 1

# d) Huffman_tree(L)
# Input:
L = [('a', '000'), ('b', '001'), ('c', '01'), ('d', '1')]
Huffman_tree(L)

# Output:
# (Huffman tree structure is constructed and stored in variable `tree`)

# e) Huffman_decode(bst, tree)
# Input:
bst = "000001001001010101111 1"
Huffman_decode(bst, tree)

# Output:
# Decoded String:
# abbcccdddd
```