Name : Harpalsinh Bhati

Enr. No.: 22162171010

Sub: Algorithm Analysis & Design

Branch: CS

Batch: 54

Sem: 5-B

Practical 10

Huffman coding assigns variable length code words to fixed length input characters based on their frequencies. More frequent characters are assigned shorter code words and less frequent characters are assigned longer code words. All edges along the path to a character contain a code digit. If they are on the left side of the tree, they will be a 0 (zero). If on the right, they'll be a 1 (one). Only the leaves will contain a letter and its frequency count. All other nodes will contain a null instead of a character, and the count of the frequency of all of it and its descendant characters.

Construct the Huffman tree for the following data and obtain its Huffman code.

Characters	**A**	**B**	**C**	**D**	**E**	**-**
Frequency/ Probability	**0.5**	**0.35**	**0.5**	**0.1**	**0.4**	**0.2**

(i) Encode text CAD-BE using the above code.

Input: CAD-BE

Output: 10011100110111100

(ii) Decode the text 1100110110 using the above information.

Input: 0011011100011100

Output: E-DAD

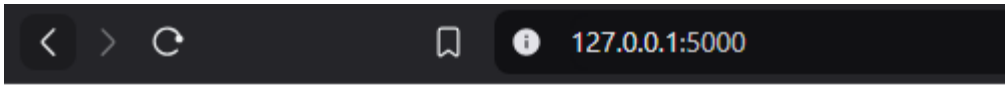
Code:

```
from flask import Flask, render_template, request
import heapq

app = Flask(__name__)
```

```python
class HuffmanNode:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        if self.freq == other.freq:
            return (self.char or '') < (other.char or '')
        return self.freq < other.freq

def build_huffman_tree(char_freq):
    heap = [HuffmanNode(char, freq) for char, freq in char_freq.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = HuffmanNode(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]

def generate_codes(root, prefix="", codes={}):
    if root is None:
        return
    if root.char is not None:
        codes[root.char] = prefix
    else:
        generate_codes(root.left, prefix + "0", codes)
        generate_codes(root.right, prefix + "1", codes)
    return codes

def encode(text, codes):
    return ''.join(codes[char] for char in text if char in codes)

def decode(binary_str, root):
    result = []
    current = root
    for bit in binary_str:
        current = current.left if bit == '0' else current.right
        if current.char is not None:
            result.append(current.char)
            current = root
    return ''.join(result)
```

```
# Character frequencies for building the Huffman Tree
char_freq = {'A': 0.5, 'B': 0.35, 'C': 0.15, 'D': 0.1, 'E': 0.4, '-': 0.2}
root = build_huffman_tree(char_freq)
codes = generate_codes(root)

@app.route("/", methods=["GET", "POST"])
def index():
    result = ""
    input_text = ""
    if request.method == "POST":
        input_text = request.form.get("input_text")
        operation = request.form.get("operation")
        if operation == "encode":
            result = encode(input_text, codes)
        elif operation == "decode":
            result = decode(input_text, root)
    return render_template("Prac_10.html", result=result,
input_text=input_text)

if __name__ == "__main__":
    app.run(debug=True)
```

Output:



Huffman Encoder/Decoder

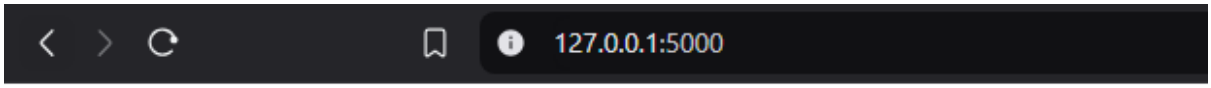
Input Text: CAD-BE

Operation: Encode

Submit

Result:

10111110101000001



Huffman Encoder/Decoder

Input Text: 011001010111010

Operation: Encode

Submit

Result:

E-DAD