Itisha Desai
Sem – 5
Branch – Cyber Security
Batch – CSE54
Enrollment No. – 22162171006

Algorithm Analysis and Design
Practical-10

Question :- 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**

1) Encode text CAD-BE using the above code.
   Input: CAD-BE
   Output: 10011100110111100
2) 0.2 Decode the text 1100110110 using the above information.
   Input: 0011011100011100
   Output: E-DAD

Code :-

app.py :

```
from flask import Flask, render_template, request, redirect, url_for, flash
import networkx as nx
import matplotlib.pyplot as plt
import os

app = Flask(__name__)
app.secret_key = 'secret_key_for_flash_messages'

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

def build_huffman_tree(char_freq):
    nodes = [Node(char, freq) for char, freq in char_freq.items()]
    while len(nodes) > 1:
        nodes = sorted(nodes, key=lambda x: x.freq)
        left = nodes.pop(0)
        right = nodes.pop(0)
        merged = Node(None, left.freq + right.freq, left, right)
        nodes.append(merged)
    return nodes[0]

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

def encode(text, codes):
    try:
        return ''.join([codes[char] for char in text])
    except KeyError:
        return "Error: Invalid character in input."

def decode(encoded_text, root):
    decoded_text = ""
    current_node = root
    for bit in encoded_text:
        current_node = current_node.left if bit == '0' else current_node.right
```

```
        if current_node.char is not None:
            decoded_text += current_node.char
            current_node = root
    return decoded_text

def plot_huffman_tree(node, graph=None, pos=None, x=0, y=0, layer=1):
    if graph is None:
        graph = nx.DiGraph()
        pos = {}
    if node:
        label = f'{node.char}:{node.freq:.2f}' if node.char else f'{node.freq:.2f}'
        graph.add_node(str(id(node)), label=label)
        pos[str(id(node))] = (x, y)
        if node.left:
            graph.add_edge(str(id(node)), str(id(node.left)))
            plot_huffman_tree(node.left, graph, pos, x - 1/layer, y - 1, layer + 1)
        if node.right:
            graph.add_edge(str(id(node)), str(id(node.right)))
            plot_huffman_tree(node.right, graph, pos, x + 1/layer, y - 1, layer + 1)
    return graph, pos

@app.route('/', methods=['GET', 'POST'])
def index():
    huffman_codes = {}
    encoded_text = ""
    decoded_text = ""
    char_freq = {}

    if request.method == 'POST':
        try:
            freq_input = request.form['char_freq'].strip().split(',')
            char_freq = {pair.split(':')[0].strip().upper(): float(pair.split(':')[1].strip()) for pair in
freq_input}

            huffman_tree = build_huffman_tree(char_freq)
            huffman_codes = generate_huffman_codes(huffman_tree)

            action = request.form['action']
            input_text = request.form['input_text'].strip().upper()

            if action == 'encode':
                encoded_text = encode(input_text, huffman_codes)
            elif action == 'decode':
                decoded_text = decode(input_text, huffman_tree)

            graph, pos = plot_huffman_tree(huffman_tree)
            labels = nx.get_node_attributes(graph, 'label')
            plt.figure(figsize=(8, 6))
```

```
        nx.draw(graph, pos, with_labels=False, node_size=2000, node_color="lightblue", font_size=10, font_weight='bold', arrows=False)
        nx.draw_networkx_labels(graph, pos, labels)
        plt.title("Huffman Tree")
        plt.savefig('static/huffman_tree.png')
        plt.close()

    except Exception as e:
        flash(f"Error: {e}. Please enter valid input.")
        return redirect(url_for('index'))

  return render_template('index.html', huffman_codes=huffman_codes, encoded_text=encoded_text, decoded_text=decoded_text)

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

index.html :

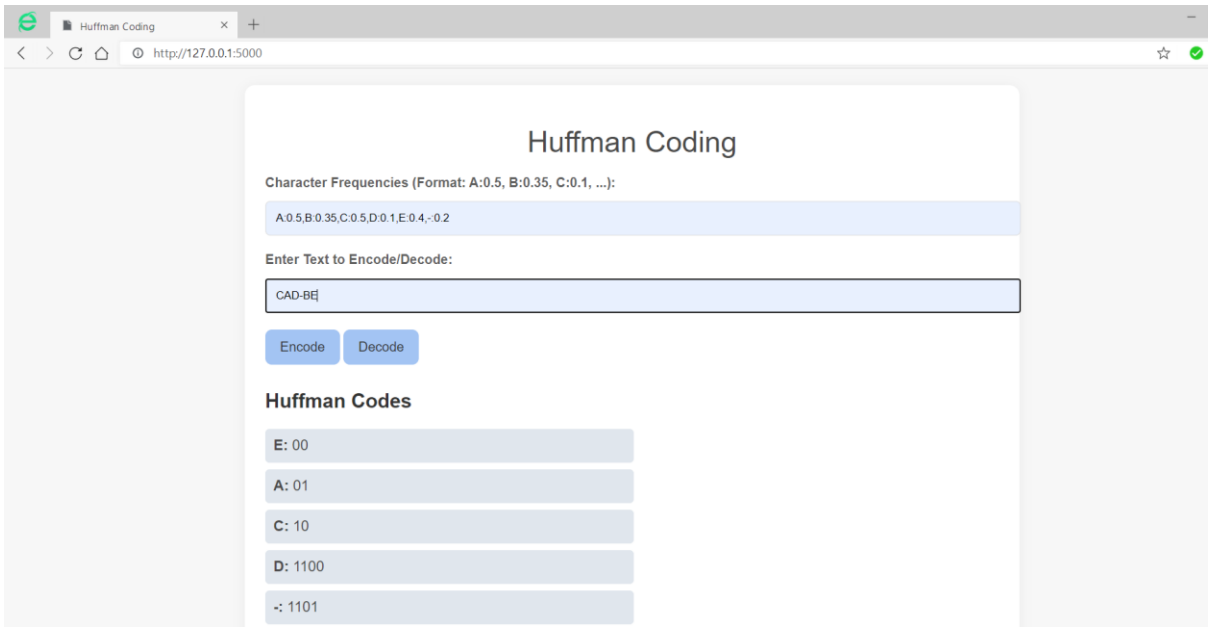
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Huffman Coding</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background-color: #f7f7f7;
      margin: 0;
      padding: 20px;
      color: #333;
    }
    h1 {
      text-align: center;
      color: #4a4a4a;
      font-size: 2.2em;
      margin-bottom: 20px;
      font-weight: 500;
    }
    .container {
      width: 60%;
      margin: 0 auto;
      background-color: #ffffff;
      padding: 25px;
      border-radius: 12px;
      box-shadow: 0 2px 15px rgba(0, 0, 0, 0.05);
```

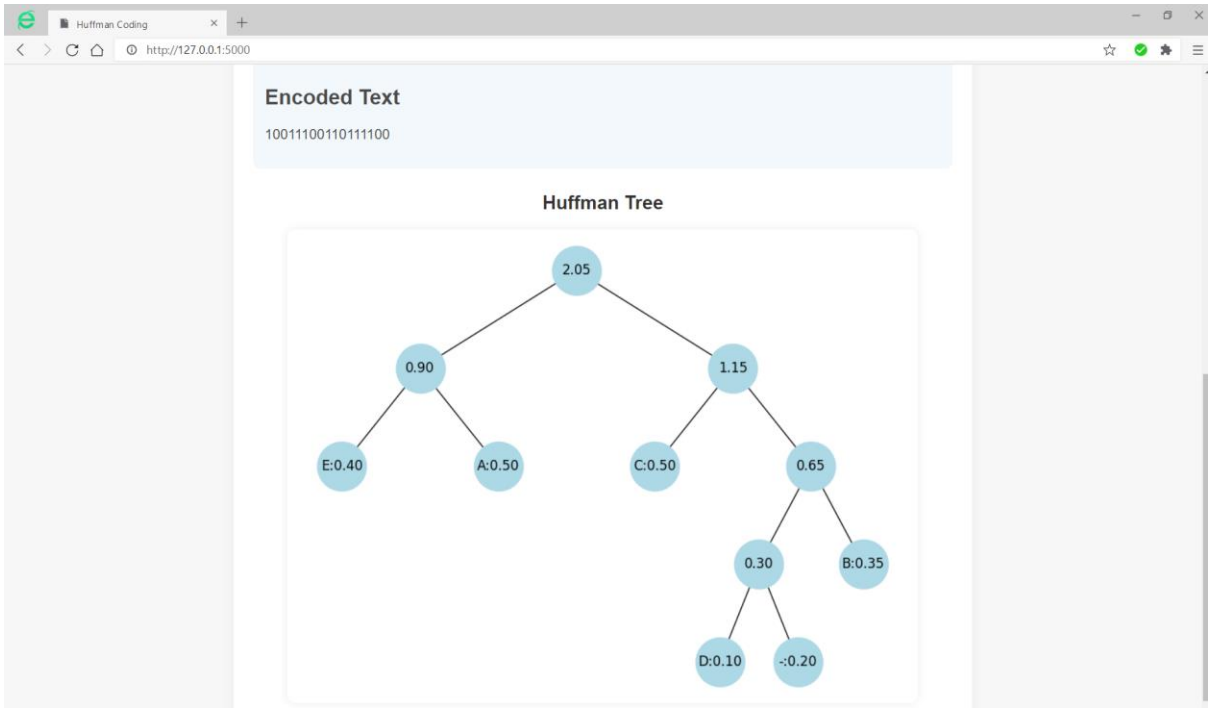
```
}
form {
    margin-bottom: 30px;
}
label {
    font-weight: bold;
    color: #5f5f5f;
    display: block;
    margin-bottom: 8px;
}
input[type="text"] {
    width: 100%;
    padding: 12px;
    margin-top: 5px;
    margin-bottom: 20px;
    border-radius: 6px;
    border: 1px solid #ddd;
    background-color: #fafafa;
}
button {
    background-color: #a3c4f3;
    color: #333;
    border: none;
    padding: 12px 18px;
    border-radius: 8px;
    cursor: pointer;
    font-size: 1em;
    transition: background-color 0.3s ease;
}
button:hover {
    background-color: #8199db;
}
.huffman-codes {
    margin-top: 30px;
}
.huffman-codes ul {
    display: flex;
    flex-wrap: wrap;
    padding: 0;
    list-style-type: none;
}
.huffman-codes li {
    flex: 0 0 48%;
    background-color: #e0e6ed;
    padding: 10px;
    margin-bottom: 8px;
    color: #4a4a4a;
    border-radius: 5px;
```

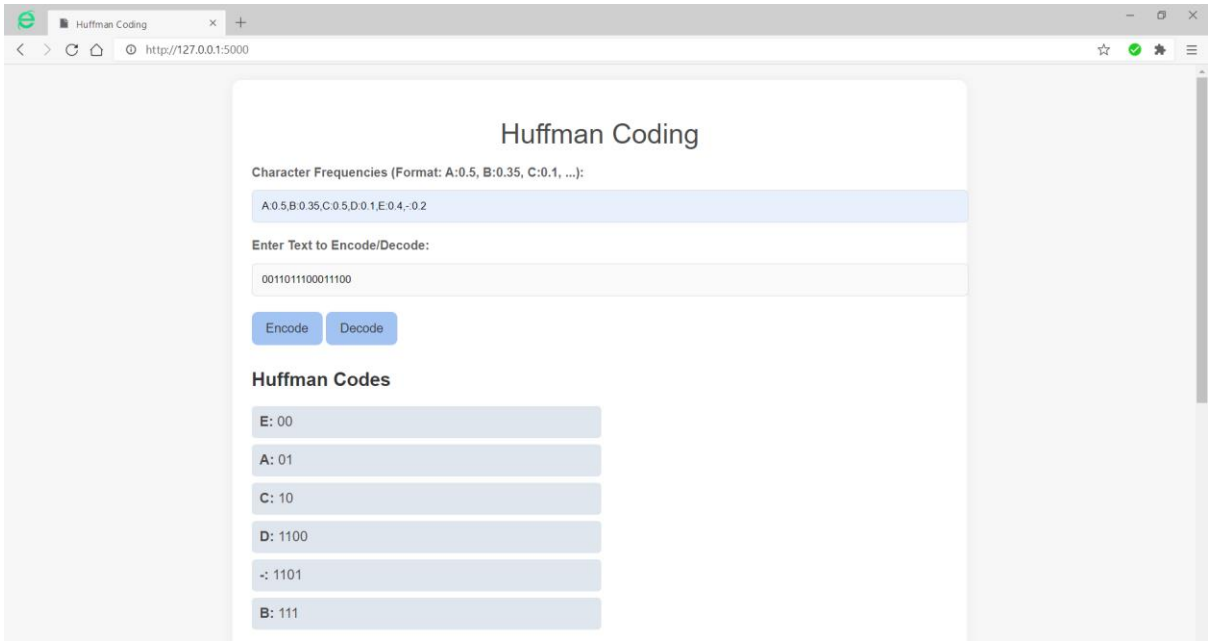
```
        margin-right: 2%;
        font-size: 1.1em;
    }
    .result {
        margin-top: 20px;
        background-color: #f3f8fc;
        padding: 15px;
        border-radius: 10px;
        color: #4a4a4a;
        font-size: 1.1em;
    }
    .error-messages ul {
        color: red;
        padding: 10px;
    }
    .huffman-tree {
        margin-top: 30px;
        text-align: center;
    }
    .huffman-tree img {
        max-width: 100%;
        height: auto;
        border-radius: 10px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.08);
    }
    @media (max-width: 768px) {
        .container {
            width: 90%;
        }
        .huffman-codes li {
            flex: 0 0 100%;
            margin-bottom: 10px;
        }
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Huffman Coding</h1>
    <form method="POST">
        <label for="char_freq">Character Frequencies (Format: A:0.5, B:0.35, C:0.1, ...):</label>
        <input type="text" id="char_freq" name="char_freq" placeholder="e.g., A:0.5, B:0.35"
required>

        <label for="input_text">Enter Text to Encode/Decode:</label>
        <input type="text" id="input_text" name="input_text" placeholder="e.g., CAD-BE" required>

        <button type="submit" name="action" value="encode">Encode</button>
```

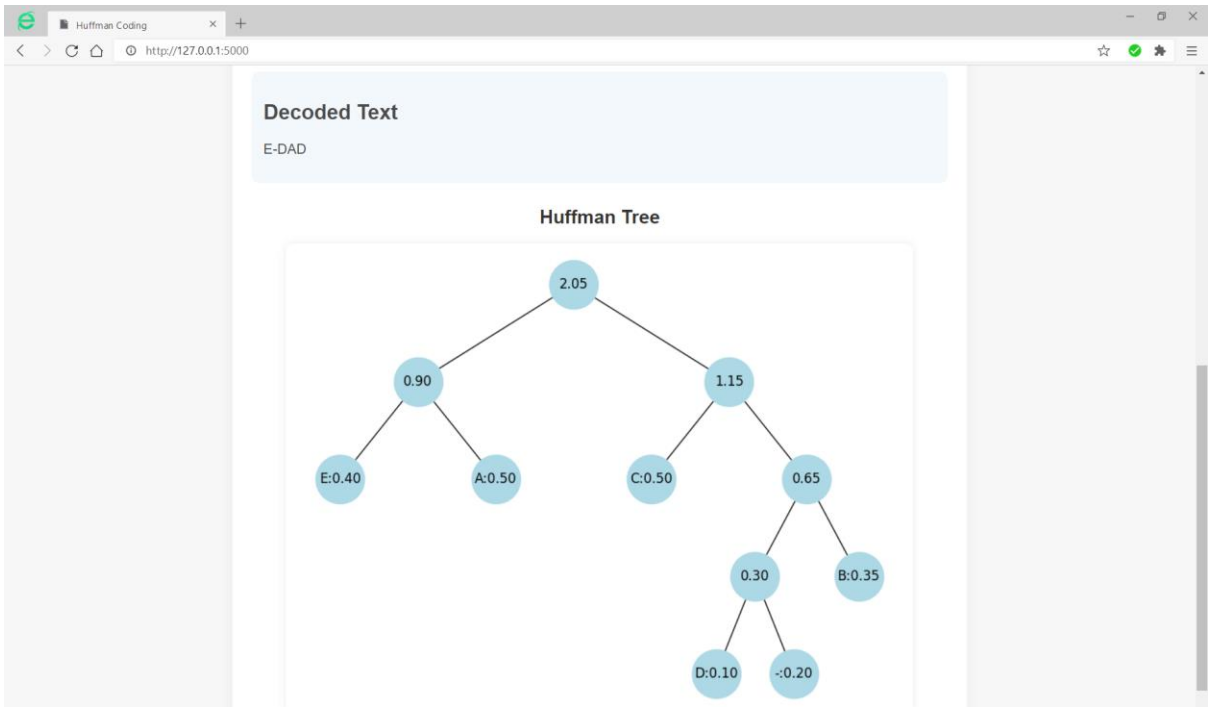
```
        <button type="submit" name="action" value="decode">Decode</button>
    </form>

    <div class="huffman-codes">
        <h2>Huffman Codes</h2>
        <ul>
            {% for char, code in huffman_codes.items() %}
            <li><strong>{{ char }}:</strong> {{ code }}</li>
            {% endfor %}
        </ul>
    </div>

    {% if encoded_text %}
    <div class="result">
        <h2>Encoded Text</h2>
        <p>{{ encoded_text }}</p>
    </div>
    {% endif %}

    {% if decoded_text %}
    <div class="result">
        <h2>Decoded Text</h2>
        <p>{{ decoded_text }}</p>
    </div>
    {% endif %}

    <div class="huffman-tree">
        <h2>Huffman Tree</h2>
        <img src="{{ url_for('static', filename='huffman_tree.png') }}" alt="Huffman Tree">
    </div>

    {% with messages = get_flashed_messages() %}
    {% if messages %}
    <div class="error-messages">
        <ul>
            {% for message in messages %}
            <li>{{ message }}</li>
            {% endfor %}
        </ul>
    </div>
    {% endif %}
    {% endwith %}
  </div>
</body>
</html>
```

Output :-


Huffman Coding
http://127.0.0.1:5000
Huffman Coding
Character Frequencies (Format: A:0.5, B:0.35, C:0.1, ...):
A:0.5,B:0.35,C:0.5,D:0.1,E:0.4,-:0.2
Enter Text to Encode/Decode:
CAD-BE
Encode
Decode
Huffman Codes
E: 00
A: 01
C: 10
D: 1100
-: 1101


Huffman Coding
http://127.0.0.1:5000
Encoded Text
10011100110111100
Huffman Tree
2.05
0.90
1.15
E:0.40
A:0.50
C:0.50
0.65
0.30
B:0.35
D:0.10
-:0.20


Huffman Coding
http://127.0.0.1:5000
Huffman Coding
Character Frequencies (Format: A:0.5, B:0.35, C:0.1, ...):
A:0.5,B:0.35,C:0.5,D:0.1,E:0.4,-:0.2
Enter Text to Encode/Decode:
0011011100011100
Encode
Decode
Huffman Codes
E: 00
A: 01
C: 10
D: 1100
-: 1101
B: 111


Huffman Coding
http://127.0.0.1:5000
Decoded Text
E-DAD
Huffman Tree
2.05
0.90
1.15
E:0.40
A:0.50
C:0.50
0.65
0.30
B:0.35
D:0.10
-:0.20