Huffman Text Compression

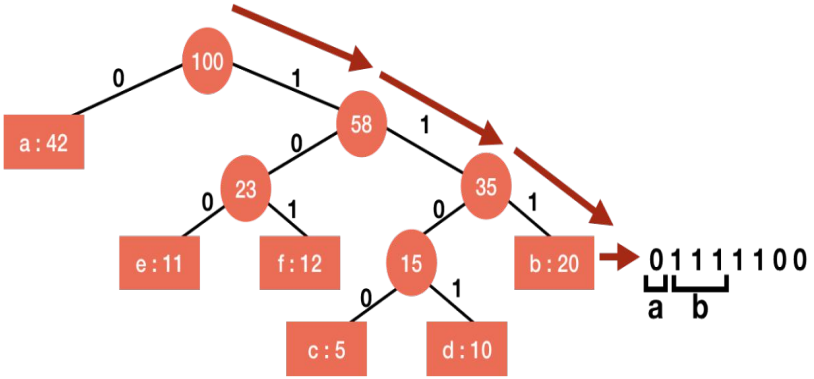
Hitesh Kumar
244161004

Algorithm and example

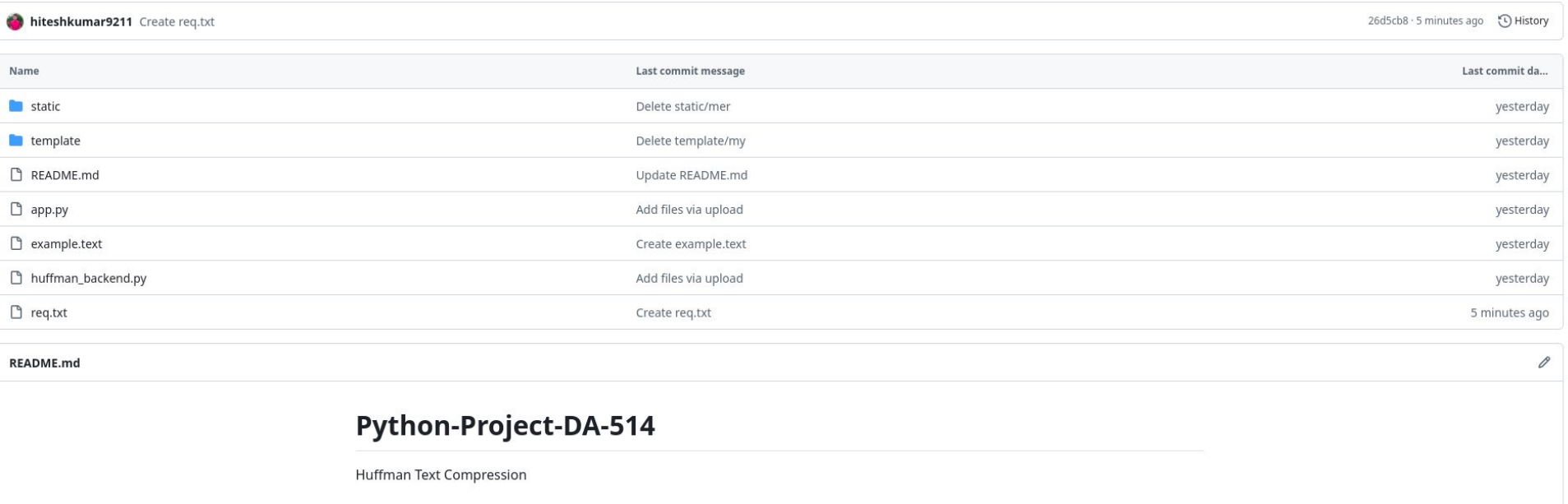
Algorithm

HUFFMAN(C)

1. *n = |C|*
2. *Q = C*
3. ***for*** *i = 1* ***to*** *n-1*
4. *allocate a new node z*
5. *z.left = x = EXTRACT-MN(Q)*
6. *z.right = y = EXTRACT-MIN(Q)*
7. *z.freq = x.freq + y.freq*
8. *INSERT(Q,z)*
9. ***return*** *EXTRACT-MIN(Q) // return the root of the tree*



Overall Structure of project.



hiteshkumar9211 Create req.txt — 26d5cb8 · 5 minutes ago — History

Name	Last commit message	Last commit da...
static	Delete static/mer	yesterday
template	Delete template/my	yesterday
README.md	Update README.md	yesterday
app.py	Add files via upload	yesterday
example.text	Create example.text	yesterday
huffman_backend.py	Add files via upload	yesterday
req.txt	Create req.txt	5 minutes ago

README.md

Python-Project-DA-514

Huffman Text Compression

huffman_backend.py

```python
from collections import Counter
import matplotlib.pyplot as plt
import networkx as nx

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

    def __lt__(self, other):
        return self.freq < other.freq

def generate_huffman_tree(text):
    text = text.replace(" ", "")
    freq_dict = Counter(text)

    nodes = [HuffmanNode(char, freq) for char, freq in freq_dict.items()]

    while len(nodes) > 1:
        nodes = sorted(nodes, key=lambda x: x.freq)
        left = nodes.pop(0)
        right = nodes.pop(0)

        merged = HuffmanNode(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        nodes.append(merged)

    root = nodes[0]
    huffman_code = {}
    build_huffman_code(root, "", huffman_code)
```

continued……

```
35
36  def build_huffman_code(node, current_code, huffman_code):
37      if node.char is not None:
38          huffman_code[node.char] = current_code
39      else:
40          build_huffman_code(node.left, current_code + "0", huffman_code)
41          build_huffman_code(node.right, current_code + "1", huffman_code)
42
43  def encode_text(text, huffman_code):
44      text = text.replace(" ", "")
45      return ''.join(huffman_code[char] for char in text)
46
47  def calculate_storage_size(encoded_text):
48      return len(encoded_text) / 8
49
50  def plot_frequency(freq_dict):
51      plt.figure(figsize=(10, 5))
52      plt.bar(freq_dict.keys(), freq_dict.values(), color="skyblue")
53      plt.xlabel('Characters')
54      plt.ylabel('Frequency')
55      plt.title('Character Frequency')
56      plot_path = 'static/frequency_plot.png'
57      plt.savefig(plot_path)
58      plt.close()
59      return plot_path
60
```

```python
def plot_huffman_tree(root):
    G = nx.DiGraph()

    pos = {}  # Position dictionary for node coordinates

    def add_edges(node, pos, parent_pos=None, x=0, y=0, level=0, x_offset=1):
        if node is not None:
            # Set the current node's position
            pos[node] = (x, y)
            G.add_node(node, label=node.char if node.char else "", weight=node.freq)
            if parent_pos is not None:
                G.add_edge(parent_pos, node)

            # Define positions for left and right child nodes
            left_x = x - x_offset / (level + 1)
            right_x = x + x_offset / (level + 1)
            y -= 1  # Move down a level

            # Recursively add edges for left and right children
            add_edges(node.left, pos, node, left_x, y, level + 1, x_offset)
            add_edges(node.right, pos, node, right_x, y, level + 1, x_offset)

    # Start adding edges from the root
    add_edges(root, pos)

    # Plot the graph
    labels = nx.get_node_attributes(G, 'label')
    plt.figure(figsize=(10, 8))
    nx.draw(G, pos, labels=labels, with_labels=True, node_size=700, node_color="skyblue", font_size=10)
    tree_path = 'static/huffman_tree.png'
    plt.savefig(tree_path)
    plt.close()
    return tree_path
```

app.py

```python
from flask import Flask, render_template, request
from huffman_backend import generate_huffman_tree, encode_text, calculate_storage_size, plot_frequency, plot_huffman_tree
import os

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    encoded_text = ""
    storage_size = 0
    plot_path = ""
    tree_path = ""
    if request.method == "POST":
        text = request.form["text"]

        # Generate Huffman tree and code
        freq_dict, huffman_code, root = generate_huffman_tree(text)
        encoded_text = encode_text(text, huffman_code)
        storage_size = calculate_storage_size(encoded_text)

        # Plot frequency and Huffman tree
        plot_path = plot_frequency(freq_dict)
        tree_path = plot_huffman_tree(root)

    return render_template("index.html", encoded_text=encoded_text, storage_size=storage_size, plot_path=plot_path, tree_path=tree_
```

```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 Text Compression</title>
    <style>
        body { font-family: Arial, sans-serif; background-color: #f4f4f9; color: #333; text-align: center; }
        h1 { color: #4a90e2; }
        form { margin-top: 20px; }
        textarea { width: 80%; height: 100px; margin-top: 10px; }
        input[type="submit"] { padding: 10px 20px; background-color: #4a90e2; color: #fff; border: none; cursor: pointer; }
        input[type="submit"]:hover { background-color: #357abd; }
        .result-section { margin-top: 30px; }
        img { margin-top: 20px; max-width: 80%; }
    </style>
</head>
<body>
    <h1>Huffman Coding Text Compression</h1>
    <form method="post">
        <label for="text">Enter Text:</label><br>
        <textarea id="text" name="text" rows="4" cols="50" placeholder="Type or paste your text here..."></textarea><br><br>
        <input type="submit" value="Compress">
    </form>

    {% if encoded_text %}
        <div class="result-section">
            <h2>Encoded Text:</h2>
            <p id="encodedText">{{ encoded_text }}</p>

            <h2>Storage Size:</h2>
            <p id="storageSize">{{ storage_size }} bytes</p>

            <h2>Frequency Plot:</h2>
            <img src="{{ plot_path }}" alt="Frequency Plot" id="frequencyPlot">

            <h2>Huffman Tree:</h2>
            <img src="{{ tree_path }}" alt="Huffman Tree" id="HuffmanTree">
        </div>
    {% endif %}

    <script>
        document.getElementById("text").addEventListener("input", function() {
            document.getElementById("encodedText").innerText = "";
            document.getElementById("storageSize").innerText = "";
            document.getElementById("frequencyPlot").style.display = "none";
            document.getElementById("HuffmanTree").style.display = "none";
        });
    </script>
</body>
</html>
```

Huffman Coding Text Compression

Enter Text:

Type or paste your text here...

Compress

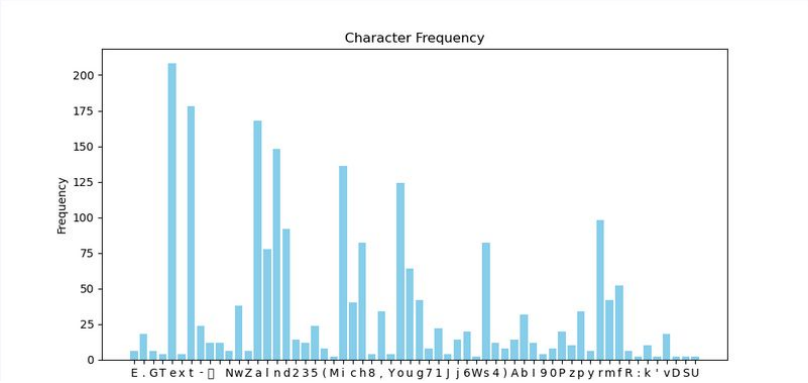
Encoded Text:

0100010010010100100010110010100101100100101111001011110000010000001000011001000110001100110010001110011101101101101011101010010000111000010010110010111011101000011111100101011100110110101100111100110101001011111011101000110011101010110011010110101100111010

Storage Size:

1323.75 bytes

Frequency Plot:



Huffman Tree: