This code implements text compression using Huffman Coding and Run-Length Encoding, along with text analysis capabilities. This task mainly focuses on the features of the Huffman Coding.

**How it works:**

1. Input text is analyzed to determine character frequencies and other properties.
2. For Huffman Coding:
    a. Analyze input text to determine character frequencies.
    b. Build a priority queue of nodes based on character frequencies.
    c. Construct a Huffman tree by repeatedly combining the two nodes with the lowest frequencies:
        i. Remove the two nodes with the lowest frequencies from the queue.
        ii. Create a new parent node with these two nodes as children.
        iii. The frequency of the parent node should be the sum of its two child nodes' frequencies.
        iv. Add the parent node back to the priority queue.
    d. Generate binary encodings for each character based on their position in the tree.
    e. Compress the text by replacing characters with their binary encodings.
    f. Decompress by traversing the Huffman tree using the compressed data.

Currently, the implementation of this feature contains a logic bug that causes the calculation of the Huffman Coding to deviate from its intended functionality.

**Expected output:**

- Compressed length: Approximately 1000-1200 bits
- Compression ratio: 50-60% of the original text length in bits
- Decompressed text: Matches the original input exactly

**Actual output:**

- Compressed length: Approximately 1400-1600 bits
- Compression ratio: 70-80% of the original text length in bits
- Decompressed text: May not match the original input for certain edge cases