Overview

This program verifies whether a message (e.g., a "ransom note") can be assembled from a given magazine by comparing the counts of required characters. The solution provides three methods for character verification, each with a unique approach to performance optimization. These methods can be benchmarked across multiple iterations to identify the most efficient solution.

Key Points Considered

1. Computational Complexity

- `Counter_method`: Uses Python's `Counter` to count characters in both the message and magazine, achieving `O(n + m)` complexity (where `n` and `m` are the lengths of the message and magazine). This approach is direct but may be less efficient for large data due to counting all characters, even unnecessary ones.

- `dict_method`: Counts characters in the magazine with a dictionary, reducing unnecessary counting by skipping repeated lookups in the message. This maintains `O(n + m)` complexity with potentially improved efficiency for non-overlapping characters. But using manually count algo can slow down execution in comparison to others methods.

- `set_method`: Utilizes `set` to check only unique characters in the message, employing `count` to validate the number of occurrences in the magazine. With early exit, it potentially improves efficiency for sparse datasets. Complexity is approximately `O(n * k)`, where `k` is the average occurrences per character but is faster in practical benchmarks due to early termination on insufficient characters.

2. Assumptions

- Input strings are expected to be text files (`message.txt` and `magazine.txt`) with alphanumeric characters and minimal punctuation. All non-alphanumeric characters are ignored.

- The program assumes files are accessible and contain sufficient data for the operations.

- If the magazine has fewer total characters than the message, the function returns `False` early to avoid unnecessary computation.

3. Outputs and Edge Cases

- Normal Output: Returns `True` if the message can be assembled from the magazine, otherwise `False`.

- Edge Cases:

  - Empty message or magazine: Returns an error message if either file is empty.

  - Insufficient magazine length: Returns `False` if the magazine is shorter than the message.

- Benchmark Mode: Outputs total execution time for each method over specified iterations, allowing performance comparison.

4. Instructions for Running

Requirements:

- The program is a Python script compatible with Python 3.6 or higher.

- Install the required `tqdm` library by running:

  pip install tqdm

Running the Program:

- The program can be executed in either normal or benchmark mode.

Command:

python ransom_note.py [message.txt] [magazine.txt] [iters]

Arguments:

- `message.txt`: Path to the text file containing the message.

- `magazine.txt`: Path to the text file containing the magazine.

- `iters` (optional): Number of iterations for benchmark mode. If omitted, the program runs in normal mode and returns `True` or `False`.

Examples:

- Normal Mode:

  python ransom_note.py message.txt magazine.txt

- Benchmark Mode (e.g., 1000 iterations):

  python ransom_note.py message.txt magazine.txt 1000