Problem 1: Find Patterns Forming Clumps in a String

1. Problem Statement

Input:

- A DNA string `Genome`.
- Three integers: `k`, `L`, and `t`.

Output:

A space-separated list of distinct `k`-mers that form `(L, t)`-clumps in the `Genome`.

Objective:

Identify all distinct `k`-mers that appear at least `t` times within any window of length `L` in the `Genome`.

Constraints:

- Length of `Genome`: 1 to 10,000.
- Integer `k`: 1 to 101.
- Integer `L`: 1 to 1,000.
- Integer `t`: 1 to 100.
- `Genome` will be composed of the characters `A`, `C`, `G`, and `T`.

#2.Algorithm Description

High-Level Idea:

Slide a window of length `L` across the `Genome`. For each window, count occurrences of every `k`-mer. If any `k`-mer appears at least `t` times, it's considered a clump. Track these clumps and return a list of distinct `k`-mers.

Pseudocode:

python-3.10

```
function find_clumps(Genome, k, L, t):
    result = []
    kmer_count = dictionary with default value 0
    found_clumps = set()
    n = len(Genome)

    if n < L or k > L:
        return []

    for i in range(n - L + 1):
        window = Genome[i:i + L]
        kmer_count.clear()

        for j in range(L - k + 1):
            kmer = window[j:j + k]
            kmer_count[kmer] += 1

        for kmer, count in kmer_count.items():
            if count >= t and kmer not in found_clumps:
                result.append(kmer)
                found_clumps.add(kmer)

    return result
```

Explanation:

The algorithm slides a window of length `L` across the `Genome` and counts the occurrences of each `k`-mer within the window. If a `k`-mer appears `t` or more times and hasn’t been recorded as a clump before, it is added to the result list.

#3.Time Analysis

Time Complexity:

- Loop through `Genome` with step `L`: `O(n - L + 1)`.
- For each window, compute all `k`-mers in that window: `O(L - k + 1)`.
- Total: `O((n - L + 1) * (L - k + 1))`.
- In the worst case, it's `O(n * L)`.

In the worst-case scenario, the complexity is approximately `O(n * L)`.

#4.Implementation

The implementation file `hw1.py` has been provided, containing the code with comments explaining each step of the algorithm.

#5.Discussion

Limitations:

- Sensitive to the values of `k`, `L`, and `t`. Adjusting these parameters can significantly affect runtime and memory usage.
- Large values for `k` or `L` can lead to high memory consumption.

Challenges:

- Accurately counting overlapping `k`-mers.
- Efficiently handling large input sizes.

Possible Improvements:

- Implementing data structures like rolling hashes to accelerate `k`-mer counting.
- Parallelizing the sliding window computation to improve performance.