Optimizing API for O(1) Time Complexity

Introduction

This section outlines the optimization strategy for fetching data from large text files efficiently, aiming for constant time complexity (`O(1)`) for API requests. The approach involves leveraging a cache system to store precomputed data.

1. Cache System Implementation

To achieve O(1) time complexity, a cache system is implemented using a map. The map stores precomputed data for specific file lines. For each file `n.txt` and line number `m`, the system creates a key in the format "n_m" and associates it with the corresponding text content.

2. Fetching Data from Cache

When a new API request is received, the system first checks the cache for the existence of the key corresponding to the requested file and line number. If the key is found, the system retrieves the data directly from the cache, avoiding the need to read the file line by line.

3. Space Complexity Consideration

While the cache system optimizes time complexity, it comes with a trade-off in space complexity. The space complexity is proportional to the product of the number of files (`n`) and the maximum number of lines (`m`) in those files. This compromise is acceptable given the potential performance gains, especially in scenarios with frequent API requests.

4. Handling Updates and New Files

To adapt to changes in the system, such as adding new files, the system periodically recomputes and updates the cache with fresh data. This ensures that the cache remains synchronized with the actual file contents and accommodates future additions.

Conclusion

The optimization approach strikes a balance between time and space complexity, providing rapid responses to API requests while allowing for system scalability and adaptability.

Handling Edge Cases for API Requests

Introduction

This section discusses the handling of edge cases to ensure the robustness and reliability of the API. Edge cases include validating file existence, checking if provided numbers are valid, and ensuring the constraints on the value of `m`.

1. Validating File Name (`n.txt`)

Before processing any API request, the system checks whether the specified file (`n.txt`) exists in the data directory. If the file is not found, the API responds with an appropriate error message, indicating that the requested file does not exist.

2. Validating Query Parameters (`n` and `m`)

The system ensures that the provided values for `n` and `m` are valid. It checks whether `n` is provided and whether `m` is a valid positive integer. If these validations fail, the API responds with a 400 Bad Request status, along with an error message indicating the issue.

3. Validating `m` Against File Length

To prevent out-of-bounds errors, the system checks if the value of `m` is within the valid range of line numbers for the specified file `n.txt`. If `m` exceeds the length of the file, the API responds with an error, indicating that the requested line number is out of bounds.