ChunkLink and LRU Cache Project Documentation

Overview

This project implements two key systems:

1. **ChunkLink System** : A file storage and reconstruction mechanism that splits files into chunks, links them using SHA-256 hashing, and ensures secure reconstruction.
2. **LRU Cache** : A Least Recently Used (LRU) caching system with O(1) time complexity for `get` and `put` operations, leveraging a doubly linked list and hash map.

The project was developed as part of a Data Structures and Algorithms course assignment, showcasing practical applications of linked lists, hashing, and algorithmic efficiency.

Team Responsibilities

- **Implementation** : Handled by Kenean Alemayhu.
- **Explanation** :
    - **ChunkLink System** : Explained by Fikir Samuel.
    - **LRU Cache** : Explained by Dagim Bireda.

ChunkLink System

Description

The **ChunkLink System** is designed to split files into manageable chunks, store them in a linked list structure, and reconstruct the file by traversing the list. Each chunk

includes metadata pointing to the next chunk, ensuring data integrity through SHA-256 checksums.

Key Features

1. **File Splitting** :
    - Files are divided into fixed-size chunks (e.g., 100KB).
    - Each chunk is stored as a node in a linked list.
2. **SHA-256 Hash-Based Chain Verification** :
    - Each chunk contains a `next_checksum`, which is the SHA-256 hash of the next chunk's data.
    - Ensures data integrity during reconstruction.
3. **Linked List Structure** :
    - Each node points to the next node, forming a chain.
4. **Secure File Reconstruction** :
    - Files are reconstructed by traversing the linked list and merging chunks.
5. **Error Handling** :
    - Detects data corruption if the `next_checksum` does not match the actual hash of the next chunk.

How It Works

1. **Node Creation** :
    - Each node stores:
        - `data`: The file chunk (bytes).
        - `next_checksum`: SHA-256 hash of the next chunk's data.
        - `next_node`: Pointer to the next node.
2. **File Splitting** :
    - The file is split into fixed-size chunks (e.g., 1MB each).
    - Metadata is generated for each chunk, including the `next_checksum`.
3. **Linked List Traversal** :
    - Start at the head node.
    - Verify the `next_checksum` matches the actual hash of the `next_node.data`.
    - If mismatch: Raise an error (data corruption detected).
4. **Reconstruction** :
    - Traverse the linked list, download chunks sequentially, and merge them to reconstruct the original file.

Deliverables

- Working code for the ChunkLink system.
- Documentation explaining the metadata format and implementation details.

LRU Cache

Description

The **LRU Cache** (Least Recently Used Cache) is a data structure that stores a limited number of items (e.g., key-value pairs) and automatically removes the least recently used item when the cache is full. It ensures O(1) time complexity for both `get` and `put` operations.

Key Features

1. **Hash Map** :
    - Stores keys and maps them to nodes in a doubly linked list for O(1) lookups.
2. **Doubly Linked List** :
    - Tracks the order of usage.
    - Most Recently Used (MRU): Items at the head of the list.
    - Least Recently Used (LRU): Items at the tail of the list.
3. **Automatic Eviction** :
    - Removes the least recently used item when the cache exceeds its capacity.
4. **Real-Time Manipulation** :
    - Add, search, or remove items interactively.

How It Works

1. **Operations** :
    - `get(key)`:
        - If the key exists:
            - Move its node to the head (mark it as "recently used").

        - Return the value.
    - If the key doesn't exist: Return `-1` or a default value.
- `put(key, value)`:
    - If the key exists:
        - Update its value.
        - Move the node to the head.
    - If the key doesn't exist:
        - Create a new node and add it to the head.
        - If the cache is full:
            - Remove the node at the tail (LRU item).
            - Remove its key from the hash map.

Deliverables

- Working code for the LRU Cache.
- Documentation explaining the metadata format and implementation details.

Technologies Used

- **Framework** : Next.js 15.1.6
- **Language** : TypeScript
- **Styling** : Tailwind CSS
- **UI Components** : shadcn/ui and Aceternity UI
- **Icons** : Lucide React
- **State Management** : React Hooks

Links to the repository

https://github.com/keneanalemayhu/chunklinkandlrucacheappllication