Advanced LRU Cache using Treaps, HashMaps, and Linked Lists

Tejas Joshi, Kaustubh Kharat, Tejas Kolhe

April 10, 2025

Abstract

The Least Recently Used (LRU) Cache is a critical component in caching mechanisms, providing efficient data retrieval while managing limited memory. In this project, we implement an advanced LRU cache using Treaps, HashMaps, and Doubly Linked Lists to optimize data access time and maintain order dynamically. This report details the problem, objectives, data structures, menu operations, and flowchart for the project.

1 Problem Statement

Efficient memory management is crucial in modern computing. Traditional LRU Cache implementations using Doubly Linked Lists and HashMaps provide fast operations but lack ordered retrieval capabilities. This project enhances the LRU Cache by incorporating a Treap, enabling ordered access while maintaining the standard O(1) `get` and `put` operations.

2 Aim

To develop an optimized LRU Cache utilizing Treaps for ordered access, HashMaps for fast lookup, and Doubly Linked Lists for efficient eviction policies.

3 Project Objectives

- Implement an LRU Cache with O(1) `get` and `put` operations.
- Integrate Treaps to maintain dynamic key ordering.
- Use HashMaps for quick key lookup.
- Utilize Doubly Linked Lists to track recently used elements.
- Develop a menu-driven C++ program for user interaction with cache.

4 Illustration of Suitable Data Structures

- **Treap**: A randomized binary search tree supporting ordered key access.
- **HashMap**: Provides O(1) time complexity for key-value retrieval and insertion.
- **Doubly Linked List**: Tracks usage order for efficient LRU eviction.

5 Menu Operations

The cache supports the following operations via a menu-driven interface:

1. **Put Key-Value Pair**: Inserts a new key or updates an existing one. If capacity is exceeded, the least recently used item is evicted.
2. **Get Value by Key**: Retrieves the value for a given key. If found, it is moved to the front (most recently used); otherwise, `-1` is returned.
3. **Delete Key**: Removes a key from the HashMap, Treap, and Linked List.
4. **Display Cache**: Outputs all key-value pairs from most to least recently used.
5. **Clear Cache**: Empties the cache completely, resetting all data structures.
6. **Resize Cache**: Allows dynamic resizing of cache capacity.
7. **Find Kth Recently Used Key**: Uses Treap structure to retrieve the k-th least recently used element.
8. **Exit**: Terminates the application.

6 Flowchart

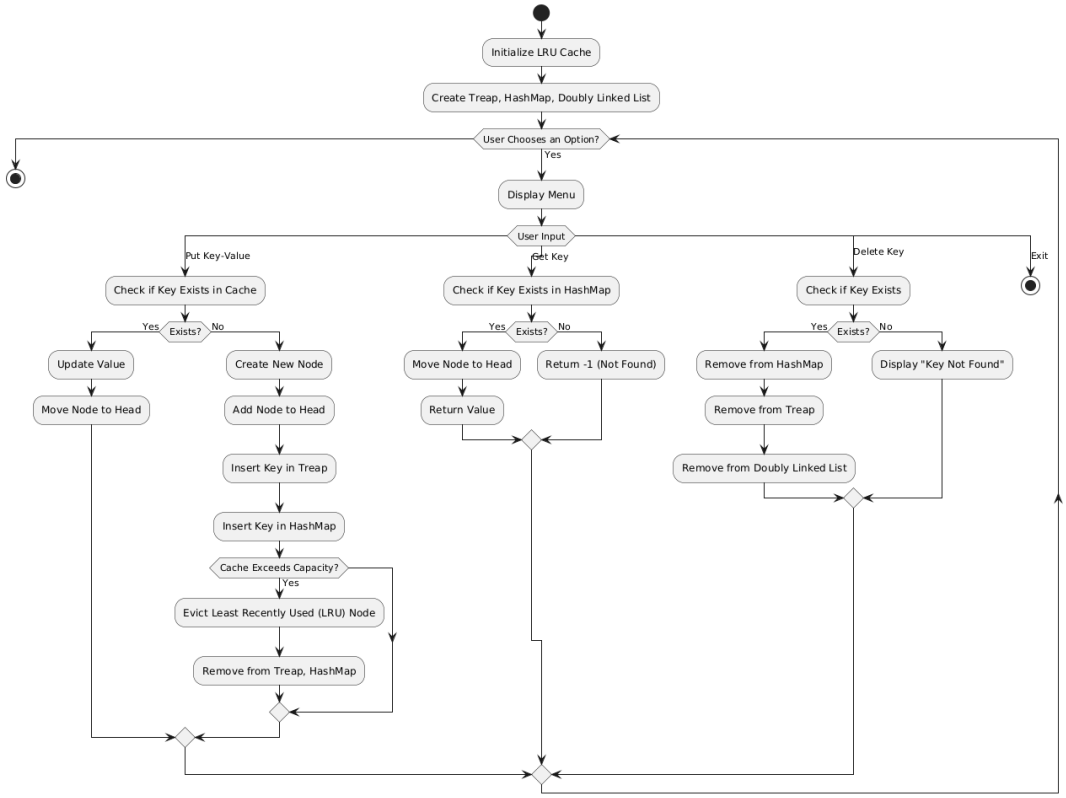

Figure 1: Basic Flowchart for Menu-driven LRU Cache using Treap, HashMap, and Doubly Linked List

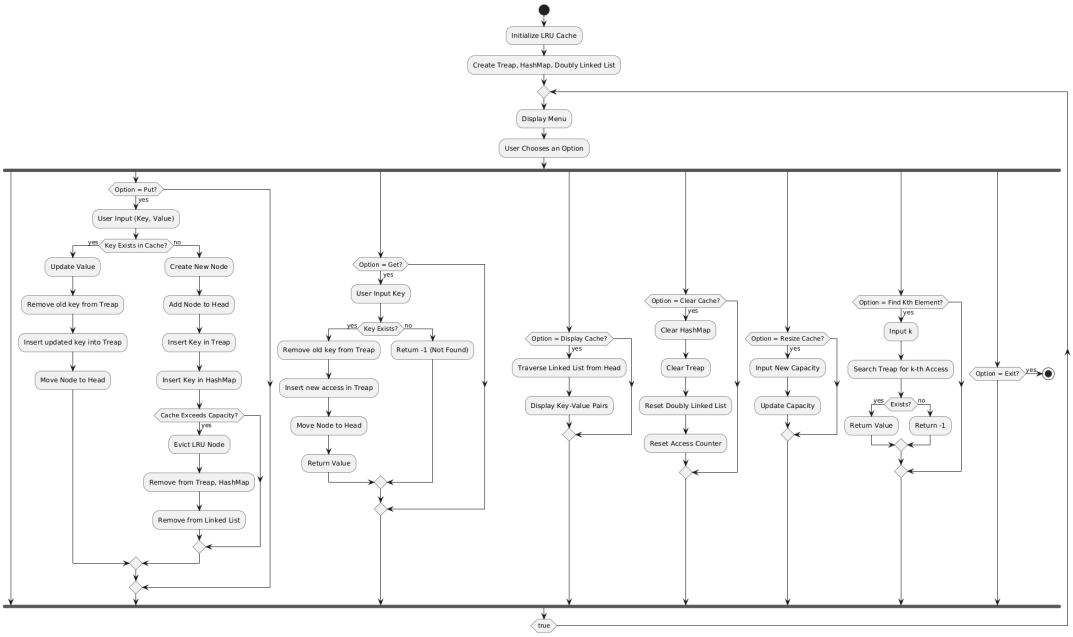

Figure 2: Complete and Final Flowchart for Menu-driven LRU Cache using Treap, HashMap, and Doubly Linked List

Each user selection is handled by branching logic, ensuring that all components update consistently. The use of Treap allows retrieval by access order (e.g., k-th recently used element), which extends beyond conventional LRU implementations.

7 Tech Stack

The technologies used in the development of the Advanced LRU Cache are as follows:

Languages

- C++ (Primary language for implementation)

Development Tools

- Visual Studio Code , Vim(Editor)
- g++ (GNU Compiler Collection)
- Git (Version Control), gdb(Debugging)
- Linux Terminal (Ubuntu)

Libraries and Dependencies

- Standard Template Library (STL) in C++ (for map, unordered_map, list)
- Custom Treap Implementation

Documentation

- LaTeX (For project report and documentation)

8 References

1. Tenenbaum, A. M. *Data Structures Using C and C++*. Pearson Education.
2. Goodrich, M. T., Tamassia, R. *Algorithm Design*. Wiley.
3. Wikipedia: Least Recently Used (LRU) Cache