NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

COMPUTER AND INFORMATION SYSTEMS ENGINEERING

DATA STRUCTURES AND ALGORITHMS (CS-218)

S.E. CIS OPEN ENDED LAB

PROJECT: LRU CACHE

GROUP MEMBERS:

AFRA KHURRAM ANSARI (CS-23008)

AMNA AHMED (CS-23016)

ZUNAIRA ZAINAB (CS-23026)

SUBMITTED TO: MISS TAHREEM KHAN

Contents

PROBLEM DESCRIPTION:

The project based on the implementation of data structures and algorithm is a project in which we have to develop a LEAST RECENTLY USED CACHE(LRU). LRU is a type of cache that maintains a limited number of item, in which the least recently used item will be evicted in the case when the cache reaches its maximum capacity. To design such structure, the operations of get and put are implemented. The project is based on a user friendly interface by adding a GUI feature that shows the cache and allows user to save the cache for later use too by saving the current cache in a database. Moreover, the track of number of hits, misses, accesses, miss rate and hit rate has also been kept.

FLOW OF PROJECT:

1. **Database Initialization and Cache Loading:** The project starts by initializing a database (cache.db) where the cache data is stored. If the cache does not already exist in the database, it is created with a table structure that holds the key, value, and an order_index to maintain the insertion order. When the application starts, the cache is populated with data from the database, ensuring that any previously stored cache data is restored.

2. **LRU Cache Data Structure:** At the core of the project is the LRU Cache class, which is built using a doubly linked list and a dictionary. This design ensures that the cache can efficiently support `O(1)` time complexity for both retrieval and insertion operations. The cache uses the following components:

   Doubly Linked List: The list maintains the order of access, with the most recentlyuse(MRU) item at the head of the list and the least recently used (LRU) items at the tail.

   Dictionary: The dictionary stores key-value pairs, allowing for fast lookups of the cache.The cache's size is limited by a defined capacity, and when the cache reaches it capacity, the least recently used item (the node at the tail of the list) is evicted.

   TIME COMPLEXITY:

   GET FUNCTION:

   Dictionary lookup: Checking if a key is in the cache (if key in self.dict) is an O(1) operation because Python dictionaries provide average constant time complexity for lookups**.**

   Node deletion and insertion: Both delete_node(node) and insert_after_node(node) methods involve manipulating the doubly linked list. Since each of these operations involves a constant number of pointer assignments (since the list is doubly linked), each of these operations takes O(1) time.

   PUT FUNCTION:

   Dictionary lookup: Checking if a key exists in the cache (if key in self.dict) is O(1).

   Node insertion and deletion: Insertion and deletion from the doubly linked list (via delete_node(node) and insert_after_node(node)) take O(1) time, as explained in the get() method**.**

   Eviction of least recently used (LRU) node: If the cache is full (i.e., the size exceeds the capacity), evicting the LRU node involves finding and removing the LRU node (done with delete_node(lru_node)), which is an O(1) operation.

So overall, get and put functions take O(1) time complexity.

SPACE COMPLEXITY:

CACHE STORAGE:

The cache uses a dictionary (self.dict) to store the keys and associated values. In the worst case, if the cache is full, it will store at most capacity entries, where each entry is a key-value pair. Therefore, the space complexity for the cache is O(capacity).

The doubly linked list also stores `capacity` nodes in the worst case, and each node stores two pointers (`left` and `right`), which also contributes O(capacity) space.

OTHER VARIABLES:

The other variables used in the code (misses, hits, accesses, dummy_head, dummy_tail) occupy constant space, O(1).

3. **Cache Operations:**

   Put Operation: When a key-value pair is added to the cache, the program checks if the key is already present. If it is, the value is updated, and the node is moved to the front of the list (marking it as the most recently used). If the key is not present, a new node is added, and if the cache is full, the least recently used item is evicted.

   Get Operation: When a key is retrieved, the program checks if the key exists in the cache. If it does, the node is moved to the front, and the value is returned. If the key is not found, the program logs a cache miss.

4. **SQLite Database Interaction:**

   - After any modification in the cache (such as a `put` operation), the cache data is saved to the database. The database is updated to reflect the current state of the cache, maintaining the correct order_index to preserve the insertion order.

   - A function is also provided to clear the database and reset the cache, which ensures that the cacHE starts fresh when needed.

5. **Miss Rate Calculation:** Throughout the operations, the program keeps track of cache misses (when the requested key is not found in the cache). The program computes the miss rate as the ratio of cache misses to the total number of accesses. This helps in evaluating the efficiency of the cache management strategy.

DISTINGUISHING FEATURES OF THE PROJECT:

GRAPHICAL USER INTERFACE: The entire project has been done using Graphical User Interface (GUI) thus making the project more user friendly and readable. The GUI is built using the Tkinter library, providing users with an interactive way to enter keys and values, retrieve values, and visualize the current state of the cache. The main GUI features are highlighted below:

- **Input Fields**: Users can enter the key and value to insert into the cache for which entry widgets have been used throughout the code.
- **Labels**: Labels have been used for displaying and readability cause.
- **Buttons**: The GUI includes buttons to perform the put and get operations, show the current cache, and save or clear the database.
- **Necessary Info**: Message box has been utilized in the project to show the info or show error in case user enters invalid input thus decreasing all the possible malfunctionality.
- **Display Table**: The current contents of the cache are displayed in a table, with items marked as either "Most Recently Used" (MRU) or "Least Recently Used" (LRU) done by using text widget.

STRUCTURED DATABASE: User can view the cache contents whenever aims of doing so. The cache contents have been divided into three sections respectively named as key, value and the status of the cache memory. This has been done to ensure a functionality for the user "Save cache for later use".The project offers the user to save the respective database whenever he desires so. Similarly, user can also clear the database if he wants to thus ensuring all the user-friendly interface for which filing has been done to do so.

PERSONALIZED THEME ADJUSTMENT: Our LRU cache offers the user two themes that is Light mode and the Dark mode theme thus user can switch between light and dark themes to enhance usability and visual appeal.

TESTING THE CACHE:

- **Show Miss/Hit Rate:** Show Miss/Hit Rate button provides the user with their full history of their cache performance including total accesses, total hits, total misses, hit ratio and miss ratio thus user has a full access to their cache performance anytime.
- **Fill cache with 50 keys:** Fill cache with 50 keys button loads users database with 50 keys starting from (0-49) and shows full info of their cache status afterwards including same options of total accesses, total hits, total misses, hit ratio and miss ratio.
- **Retrieve Odd keys:** User can retrieve odd keys whenever from their cache contents which actually does load the database with odd keys and shows full info of their cache database afterwards**.**
- **Put Primes 1 till 100:** Then the keys are put from 1 till 100 for such numbers which are prime and user can also view full cache status for it through the button of show miss/hit rate to see the cache final miss rate at the end.

MOST CHALLENGING PART OF THE PROJECT:

ORGANIZING THE DATABASE: Organizing the database was indeed the hardest and the most challenging part of the project since it came with the following challenges and difficulties:

- Keeping the in-memory cache and database consistent during additions, evictions, and updates.
- Avoiding performance issues caused by frequent database writes while ensuring reliability.
- Managing errors like missing or corrupted database files gracefully without crashing the program.
- Implementing a reset functionality that clears both the database and the in-memory cache while updating the GUI accordingly.

PERSONALIZING THE THEME:

- **Consistency Across Components**: Ensuring that the colors, font styles, and overall aesthetics are applied uniformly across all GUI elements (labels, buttons, entries, treeviews, etc.).
- **Dynamic Theme Switching**: Implementing a seamless toggle mechanism for light and dark themes without visual glitches or the need to restart the application.
- **Customization Flexibility**: Adapting the theme functions to accommodate additional customizations (e.g., user-defined colors or fonts) beyond the predefined themes.
- **Treeview Styling Complexity**: Managing the unique styling of Treeview elements, especially with tags for MRU and LRU, which require separate configuration for background colors and highlights.
- **Preserving Accessibility**: Ensuring readability and usability of the interface for all users, including those with visual impairments, by maintaining proper contrast ratios and font sizes.

NEW LEARNINGS DURING THE PROJECT:

1. **Sqlite3 Module**: The sqlite3 module in Python provides a lightweight and built-in interface to work with SQLite databases. It enables the creation, modification, and querying of databases directly from Python scripts without needing an external database server. This was new to learn and essential for implementing persistent storage in the project.
2. **ttk Module:** The ttk module (part of the tkinter library) offers a modernized set of widgets for building graphical user interfaces (GUIs). Unlike standard tkinter widgets, ttk widgets provide a more visually appealing and customizable interface. Learning how to use ttk. Treeview to display cache data in a tabular format was a new and valuable addition. Customizing the appearance and functionality of a Treeview widget in GUI frameworks like provides a default table-like interface for displaying hierarchical or tabular data.

CONTRIBUTION OF GROUP MEMBERS:

MEMBER 1: AFRA

Afra contributed to the project by implementing the core functionality of the project through the LRU Class implementation and contributed in customizing the main GUI window for theme toggling of light to dark and dark to light themes.

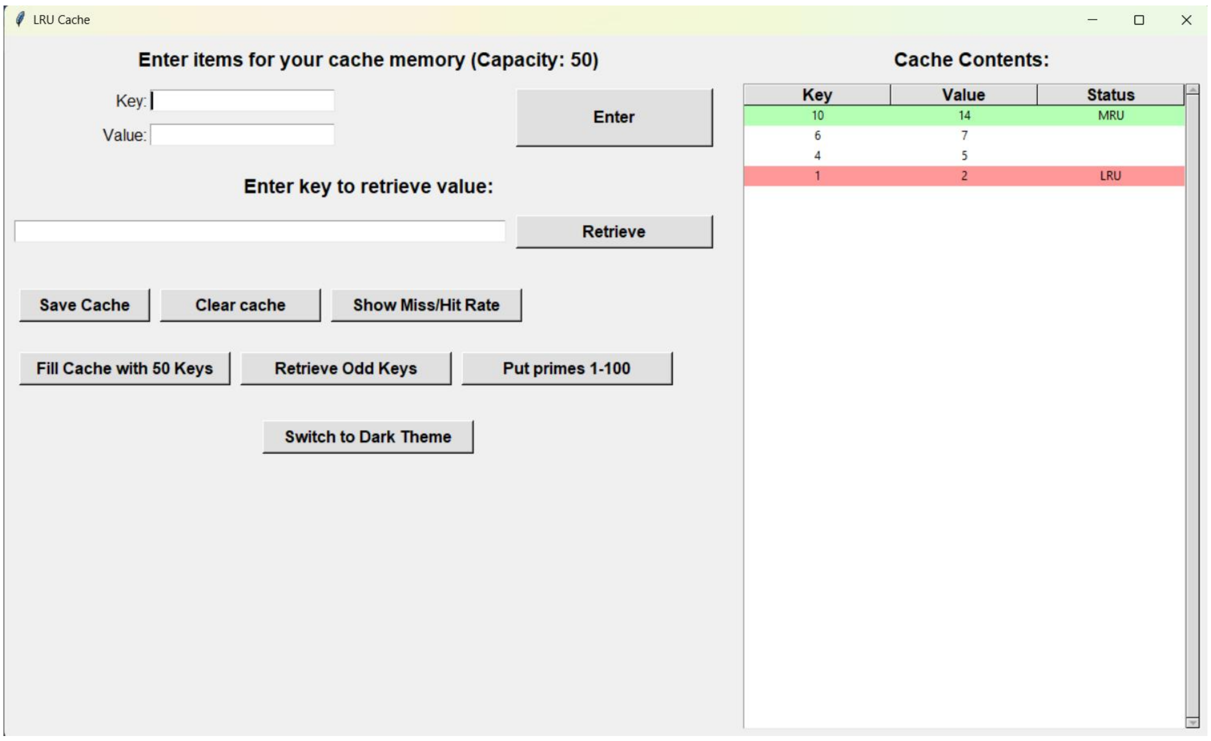
MEMBER 2: ZUNAIRA

Zunaira implemented the Graphical User Interface using all the GUI features and functionalities effectively.

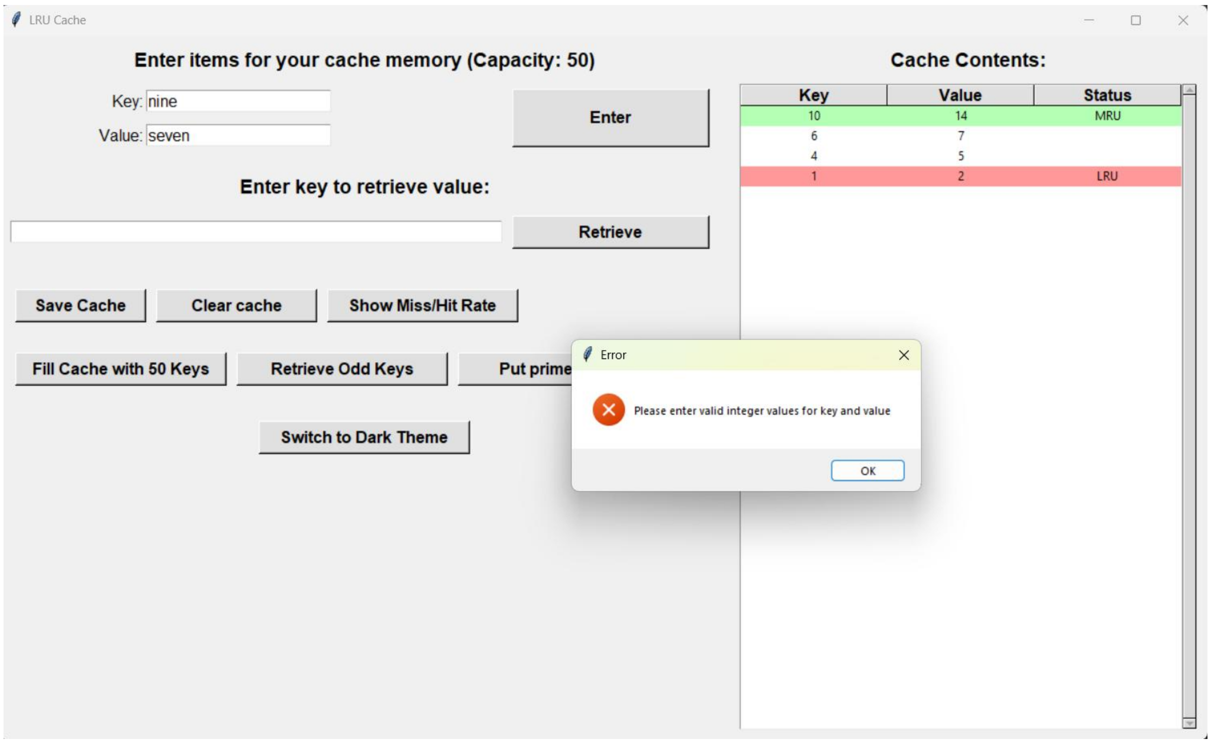
MEMBER 3: AMNA

Amna focused on the the database handling part of the project effectively highlighting the MRU and LRU, updating the database, clearing it and saving the database.

SCREENSHOTS:



Key	Value	Status
10	14	MRU
6	7	
4	5	
1	2	LRU



Enter items for your cache memory (Capacity: 50)

Key: nine

Value: seven

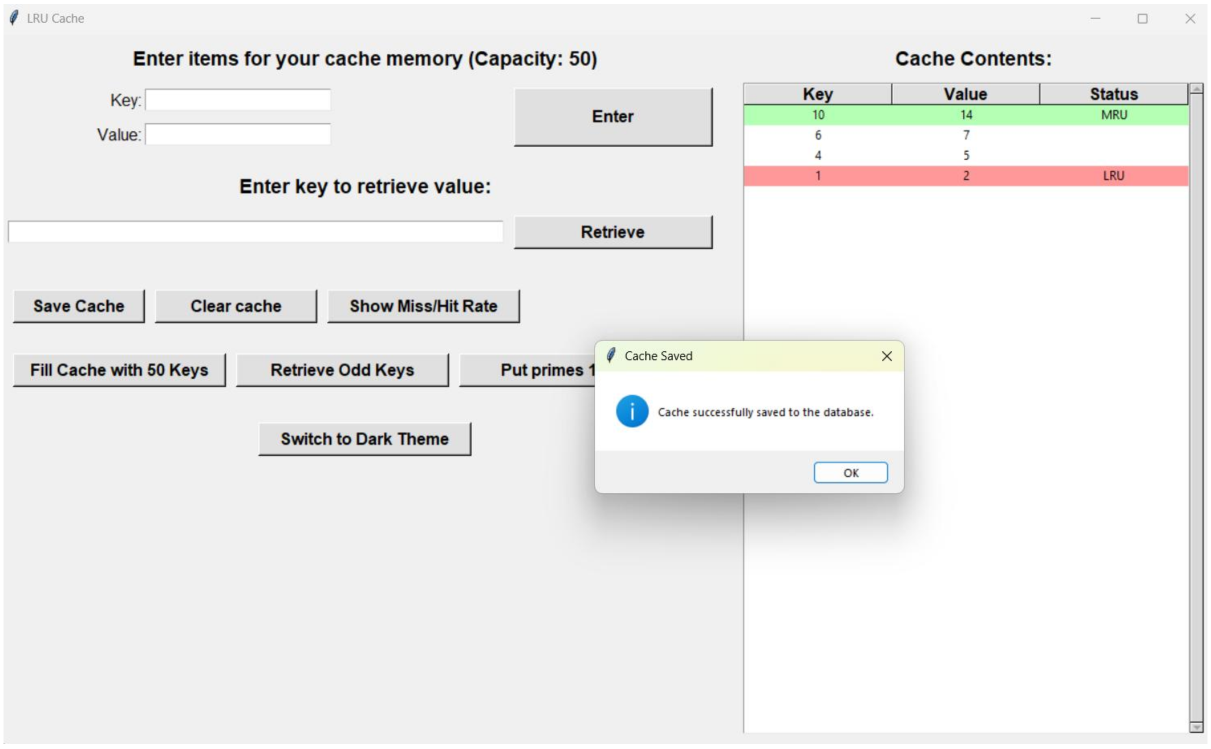
Enter key to retrieve value:

Cache Contents:

Key	Value	Status
10	14	MRU
6	7	
4	5	
1	2	LRU

Error

Please enter valid integer values for key and value



Enter items for your cache memory (Capacity: 50)

Key:

Value:

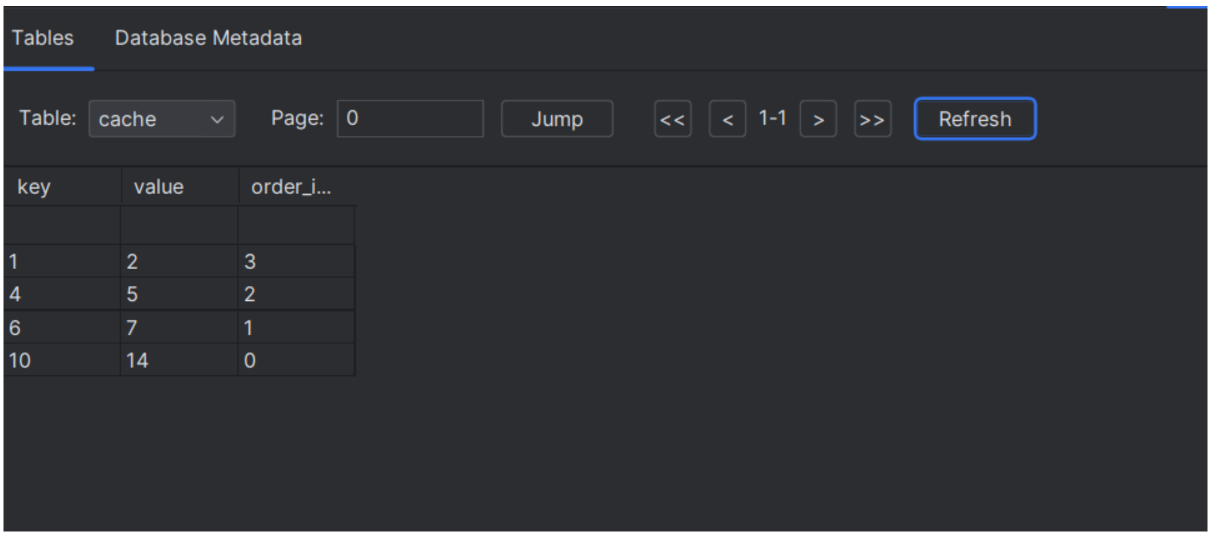
Enter key to retrieve value:

Cache Contents:

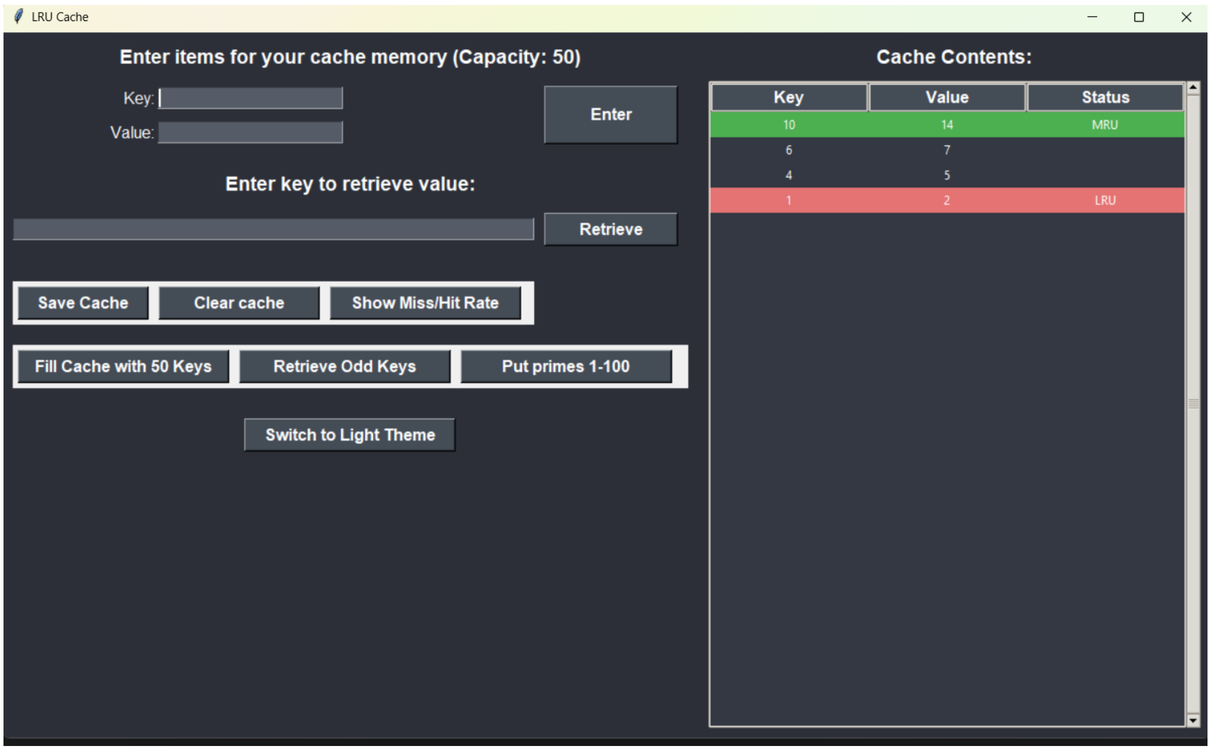
Key	Value	Status
10	14	MRU
6	7	
4	5	
1	2	LRU

Cache Saved

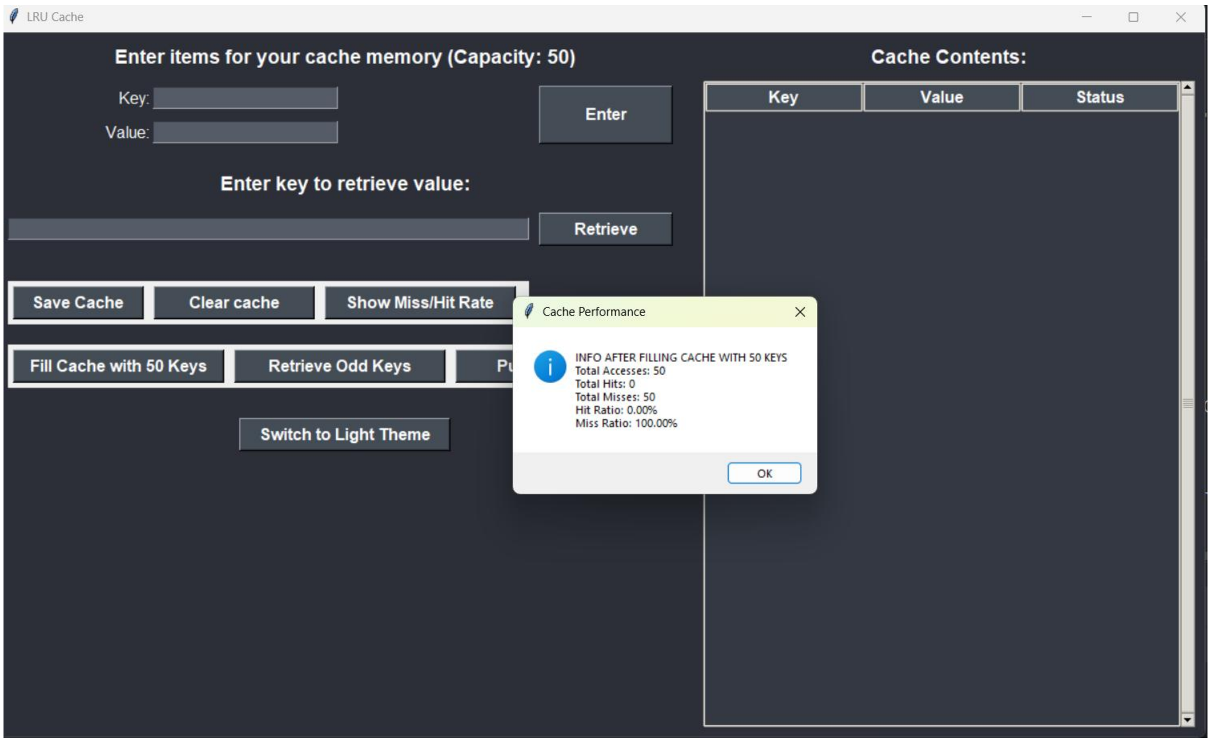
Cache successfully saved to the database.



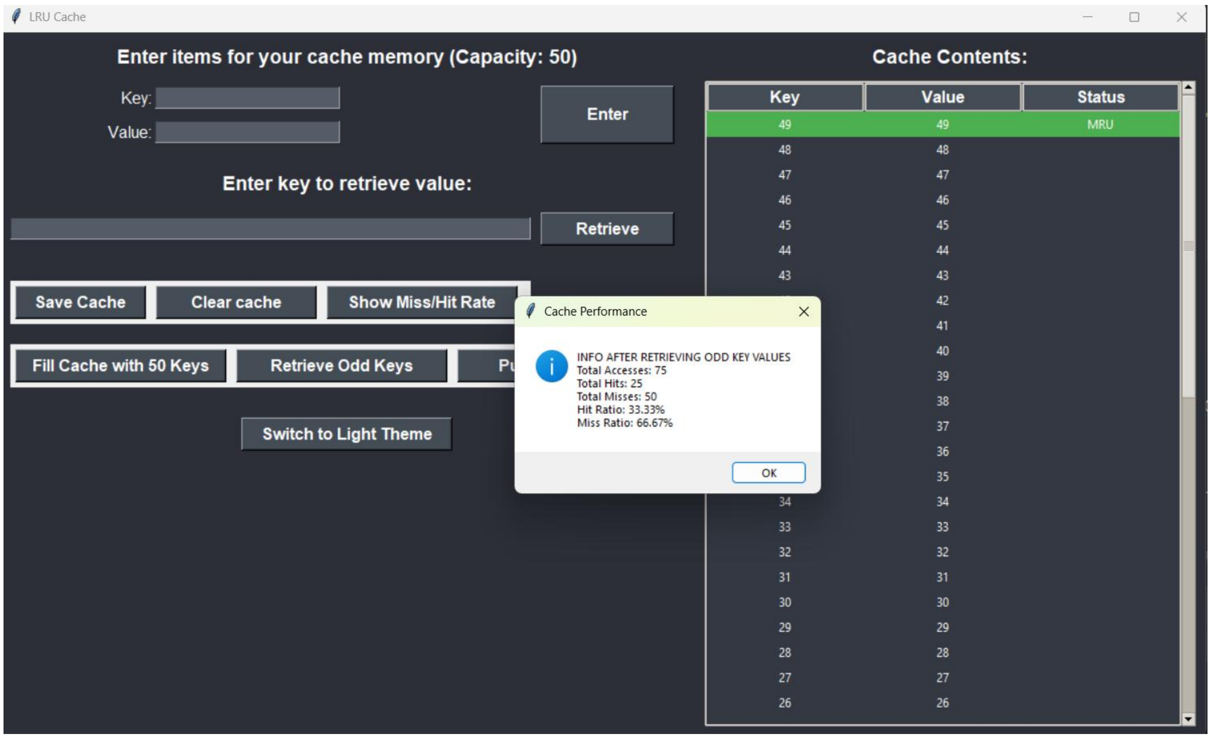
key	value	order_i...
1	2	3
4	5	2
6	7	1
10	14	0



Key	Value	Status
10	14	MRU
6	7	
4	5	
1	2	LRU



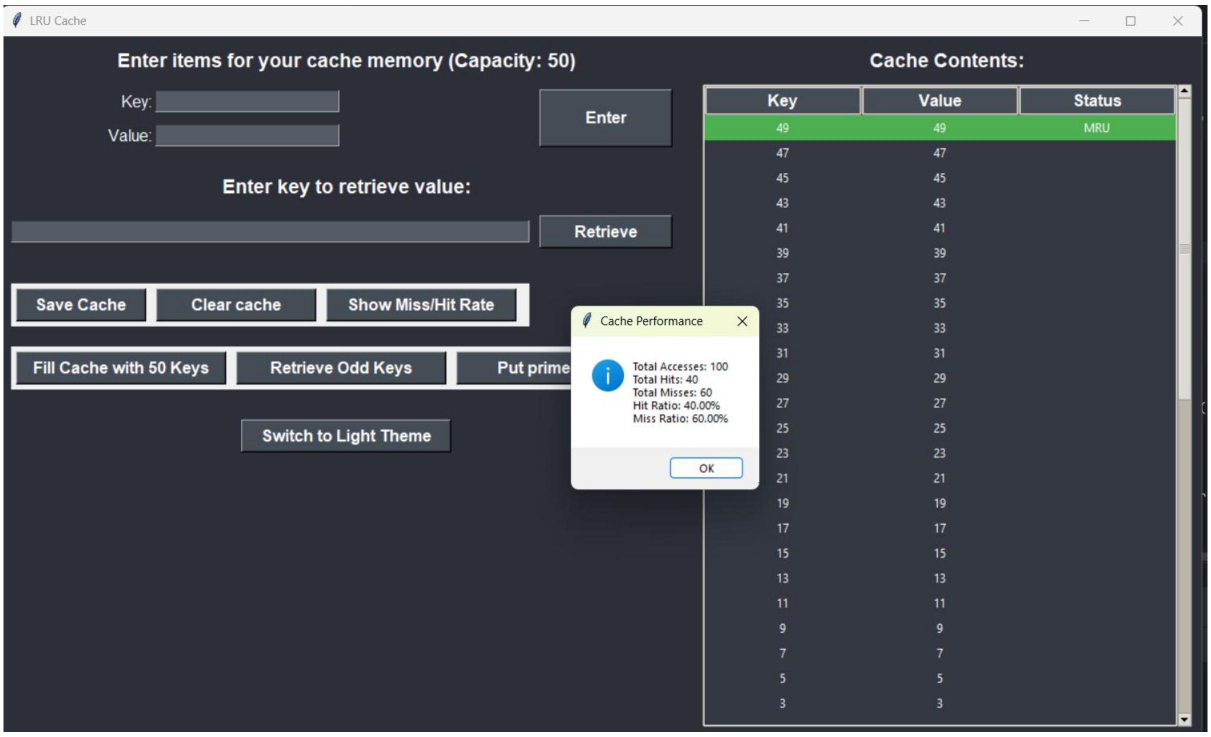
INFO AFTER FILLING CACHE WITH 50 KEYS
Total Accesses: 50
Total Hits: 0
Total Misses: 50
Hit Ratio: 0.00%
Miss Ratio: 100.00%



INFO AFTER RETRIEVING ODD KEY VALUES
Total Accesses: 75
Total Hits: 25
Total Misses: 50
Hit Ratio: 33.33%
Miss Ratio: 66.67%

Key	Value	Status
49	49	MRU
48	48	
47	47	
46	46	
45	45	
44	44	
43	43	
	42	
	41	
	40	
	39	
	38	
	37	
	36	
	35	
34	34	
33	33	
32	32	
31	31	
30	30	
29	29	
28	28	
27	27	
26	26	

FINAL MISS RATE:



Total Accesses: 100
Total Hits: 40
Total Misses: 60
Hit Ratio: 40.00%
Miss Ratio: 60.00%

Key	Value	Status
49	49	MRU
47	47	
45	45	
43	43	
41	41	
39	39	
37	37	
35	35	
33	33	
31	31	
29	29	
27	27	
25	25	
23	23	
21	21	
19	19	
17	17	
15	15	
13	13	
11	11	
9	9	
7	7	
5	5	
3	3	