# LRU Cache

The objective of an LRU cache is to have rapid access to data in finite storage. There must be a get() and set() where the getter will have to access a value in O(1) time. On top of that, insertion and deletion must also be constant time.

I implemented the LRU cache with a doubly linked list that works similar to a queue but it adds values from the front and removes from the back. One of the main functions of an LRU cache is inserting from the front and evicting from the back, linked lists only point to the next node in memory but having a tail for the doubly linked list, allows constant time eviction. To get constant time insertion and deletion, and access to any item in the middle, I used a dictionary. With the dictionary, I can keep track of the node and the key; the reason I stored the node instead of its value in it is so that I can have that constant time access. As for space complexity, it uses a dictionary and a doubly linked list making it O(n).

## Sources


ayushjauhari14. (2022, October 20). *Implementation of deque using doubly linked list*. GeeksforGeeks. From https://www.geeksforgeeks.org/implementation-deque-using-doubly-linked-list/

Back To Back SWE. (2019). *Implement An LRU Cache - The LRU Cache Eviction Policy ("LRU Cache" on LeetCode)*. *YouTube*. From https://www.youtube.com/watch?v=S6IfqDXWa10

Berestovskyy, A. (2018, April 9). *Cache performance in hash tables with chaining vs open addressing*. Stack Overflow. From https://stackoverflow.com/questions/49709873/cache-performance-in-hash-tables-with-chaining-vs-open-addressing

paxdiablo. (2011, May 28). *Time complexity of doubly linked list element removal?* Stack Overflow. From https://stackoverflow.com/questions/6161615/time-complexity-of-doubly-linked-list-element-removal