Problem List Submit 0 Premium

Description Editorial Solutions Submissions

146. LRU Cache

Solved

Medium Topics

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `-1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

The functions `get` and `put` must each run in `O(1)` average time complexity.

Example 1:

Input
`["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]`
`[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]`

Code

Python3 Auto

```python
class LRUCache:

    class Node:
        def __init__(self, key, value):
            self.key = key
            self.value = value
            self.prev = None
            self.next = None

    def __init__(self, capacity: int):
        self.capacity = capacity
        self.cache = {}  # key -> Node

        # Dummy head and tail (sentinels)
```

Restored from local Upgrade to Cloud Saving Ln 1, Col 1

Testcase Test Result

Case 1

```
["LRUCache","put","put","get","put","get","put"
,"get","get","get"]
```

```
[[2],[1,1],[2,2],[1],[3,3],[2],[4,4],[1],[3],
[4]]
```