Implement LRU Cache

Try to solve the Implement Least Recently Used (LRU) cache problem.

We'll cover the following

Statement

Implement an LRU cache class with the following functions:

- **Init(capacity)**: Initializes an LRU cache with the **capacity** size.
- **Set(key, value)**: Adds a new key-value pair or updates an existing **key** with a new **value**.
- **Get(key)**: Returns the value of the **key**, or -1 if the **key** does not exist.

If the number of keys has reached the cache **capacity**, evict the least recently used **key** and then add the new **key**.

As caches use relatively expensive, faster memory, they are not designed to store very large data sets. Whenever the cache becomes full, we need to evict some data from it. There are several caching algorithms to implement a cache eviction policy. LRU is a very simple and commonly used algorithm. The core concept of the LRU algorithm is to evict the oldest data from the cache to accommodate more data.

Constraints:

- $1 \leq$ **capacity** ≤ 3000
- $0 \leq$ **key** $\leq 10^4$
- $0 \leq$ **value** $\leq 10^5$
- At most 2×10^5 calls will be made to **Set** and **Get**.

Examples

Are you sure you want to see solution?

YES CANCEL

Understand the problem

Let’s take a moment to make sure you've correctly understood the problem. The quiz below helps you check if you're solving the correct problem:

Implement LRU Cache

1

Suppose we have a cache with a capacity of 4. What is the output if we set a new pair with the following inputs?

key = 15

value = 100

key	value
17	25
23	17
12	31

A)

key	value
15	100
23	17
17	25
12	31

B)

key	value
17	25
15	100
23	17
12	31

C)

key	value
17	25
23	17
12	31
15	100

D)

key	value
17	25
23	17
15	100
12	31

Submit Answer

Question 1 of 3
0 attempted

Reset Quiz

Figure it out!

We have a game for you to play. Rearrange the logical building blocks to develop a clearer understanding of how to solve this problem.

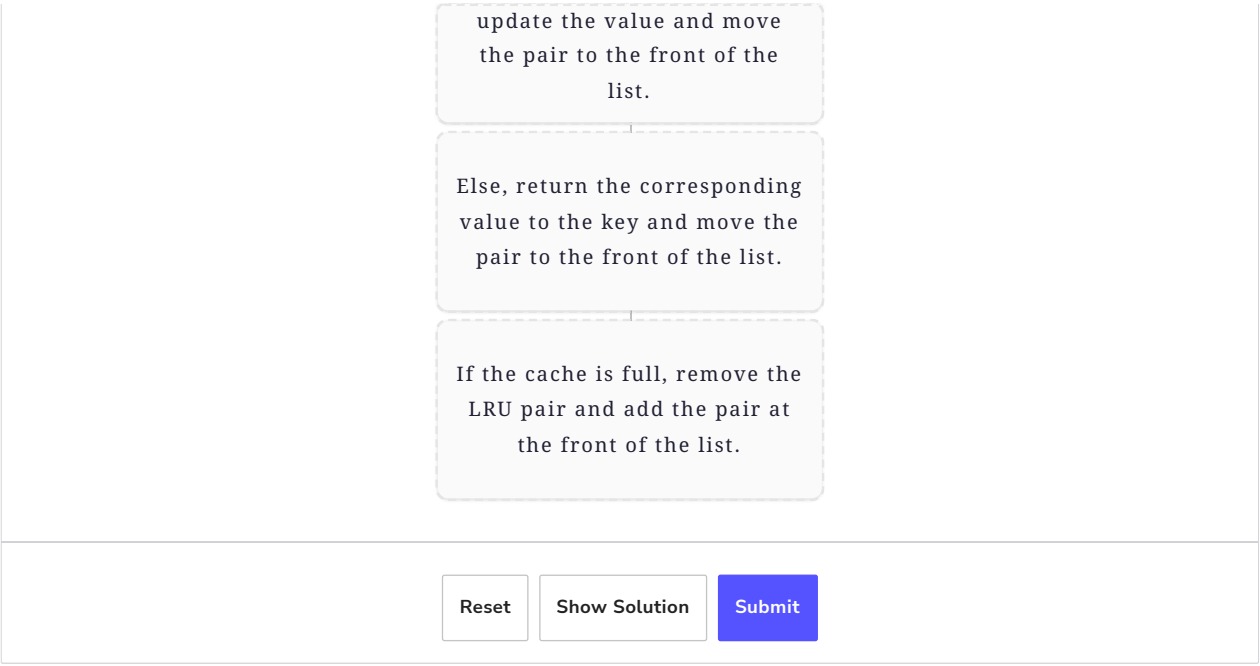
> **Note:** Focus on setting the value and then getting the value.

Drag and drop the cards to rearrange them in the correct sequence.

To get a value if the given key doesn't exist, return -1.

If the key doesn't exist, check whether the cache is full. If there's the capacity to add a new pair, then add it at the

To set a pair, if the given key already exists, then we'll



Try it yourself

Implement your solution in `ImplementLRUCache.java` in the following coding playground. You'll need the provided supporting code to implement your solution.

Java

ImplementLRUCache.java

LinkedListNode.java

LinkedList.java

```
1  import java.util.*;
2
3  class LRUCache {
4      // Constructor that sets the size of the cache
5      public LRUCache(int size) {
6          // Write your code here
7      }
8
9      int get(int key) {
10         // Your code will replace this placeholder return statement
11
```