# First Unique Number

Solution

You have a queue of integers, you need to retrieve the first unique integer in the queue.

Implement the `FirstUnique` class:

- `FirstUnique(int[] nums)` Initializes the object with the numbers in the queue.
- `int showFirstUnique()` returns the value of **the first unique** integer of the queue, and returns **-1** if there is no such integer.
- `void add(int value)` insert value to the queue.

**Example 1:**

```
Input:
["FirstUnique","showFirstUnique","add","showFirstUnique","add","showFirstUnique","add","showFirstUnique"]
[[[2,3,5]],[],[5],[],[2],[],[3],[]]
Output:
[null,2,null,2,null,3,null,-1]

Explanation:
FirstUnique firstUnique = new FirstUnique([2,3,5]);
firstUnique.showFirstUnique(); // return 2
firstUnique.add(5);            // the queue is now [2,3,5,5]
firstUnique.showFirstUnique(); // return 2
firstUnique.add(2);            // the queue is now [2,3,5,5,2]
firstUnique.showFirstUnique(); // return 3
firstUnique.add(3);            // the queue is now [2,3,5,5,2,3]
firstUnique.showFirstUnique(); // return -1
```

**Example 2:**

```
Input:
["FirstUnique","showFirstUnique","add","add","add","add","add","showFirstUnique"]
[[[7,7,7,7,7,7]],[],[7],[3],[3],[7],[17],[]]
Output:
[null,-1,null,null,null,null,null,17]

Explanation:
FirstUnique firstUnique = new FirstUnique([7,7,7,7,7,7]);
firstUnique.showFirstUnique(); // return -1
firstUnique.add(7);            // the queue is now [7,7,7,7,7,7,7]
firstUnique.add(3);            // the queue is now [7,7,7,7,7,7,7,3]
firstUnique.add(3);            // the queue is now [7,7,7,7,7,7,7,3,3]
firstUnique.add(7);            // the queue is now [7,7,7,7,7,7,7,3,3,7]
firstUnique.add(17);           // the queue is now [7,7,7,7,7,7,7,3,3,7,17]
firstUnique.showFirstUnique(); // return 17
```

**Example 3:**

```
Input:
["FirstUnique","showFirstUnique","add","showFirstUnique"]
[[[809]],[],[809],[]]
Output:
[null,809,null,-1]

Explanation:
FirstUnique firstUnique = new FirstUnique([809]);
firstUnique.showFirstUnique(); // return 809
firstUnique.add(809);          // the queue is now [809,809]
firstUnique.showFirstUnique(); // return -1
```

**Constraints:**

- `1 <= nums.length <= 10^5`
- `1 <= nums[i] <= 10^8`
- `1 <= value <= 10^8`

- At most `50000` calls will be made to `showFirstUnique` and `add`.

Hide Hint #1

Use doubly Linked list with hashmap of pointers to linked list nodes. add unique number to the linked list. When add is called check if the added number is unique then it have to be added to the linked list and if it is repeated remove it from the linked list if exists. When showFirstUnique is called retrieve the head of the linked list.

Hide Hint #2

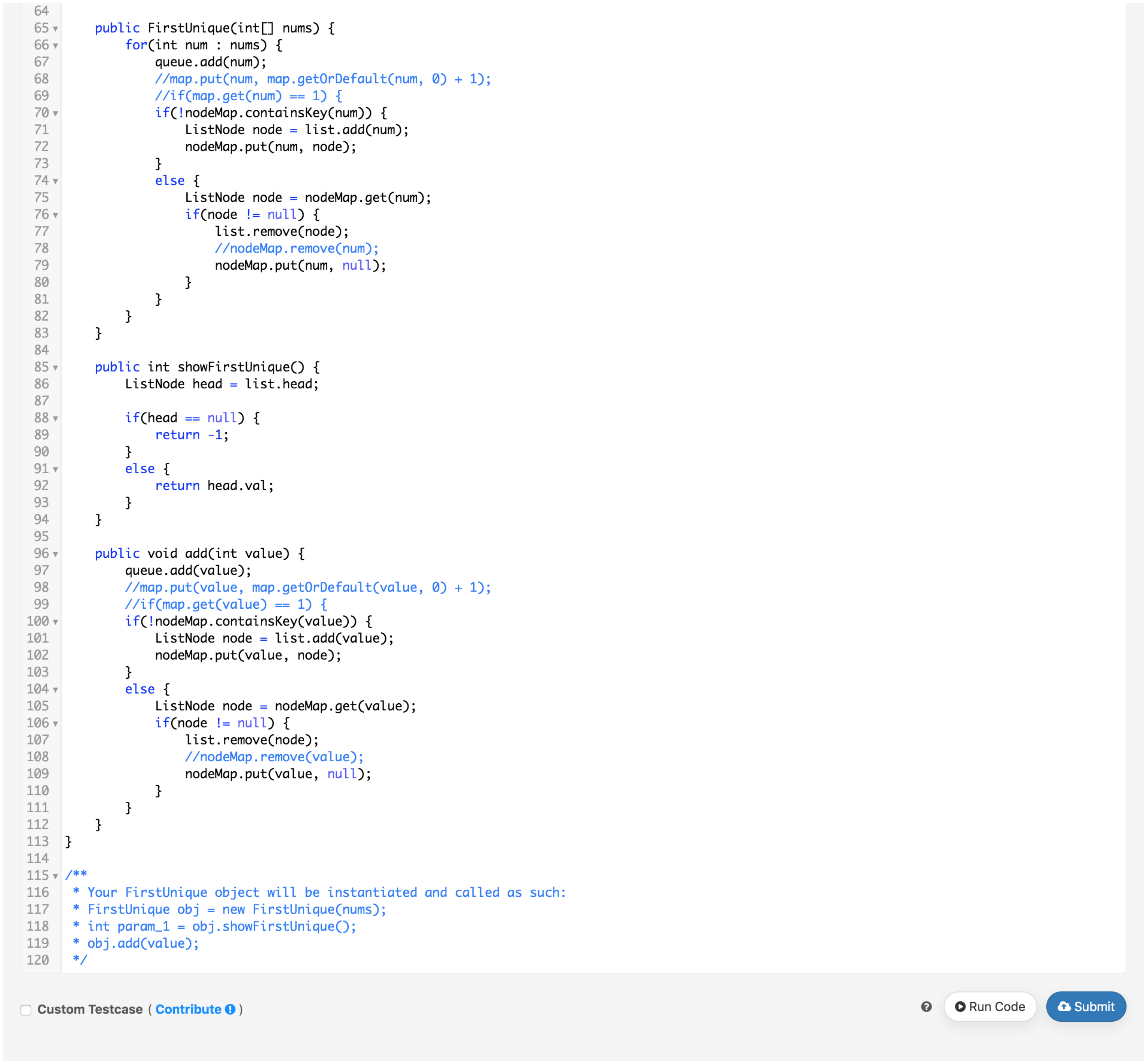
Use queue and check that first element of the queue is always unique.

Hide Hint #3

Use set or heap to make running time of each function O(logn).

Java

```java
class FirstUnique {

    private class ListNode {
        int val;
        ListNode next;
        ListNode prev;

        public ListNode(int value) {
            val = value;
            next = null;
            prev = null;
        }
    }

    private class LinkedListCustome {
        ListNode head;
        ListNode tail;

        public LinkedListCustome() {
            head = null;
            tail = null;
        }

        public ListNode add(int value) {
            ListNode node = new ListNode(value);
            if(head == null) {
                head = node;
                tail = node;
            }
            else {
                tail.next = node;
                node.prev = tail;
                tail = tail.next;
            }

            return node;
        }

        public void remove(ListNode node) {
            if(node == tail && node == head) {
                tail = null;
                head = null;
            }
            else if(node == tail) {
                tail = tail.prev;
                tail.next = null;
            }
            else if(node == head){
                head = head.next;
                head.prev = null;
            }
            else {
                node.prev.next = node.next;
                node.next.prev = node.prev;
            }
        }
    }


    Queue<Integer> queue = new LinkedList();
    //HashMap<Integer, Integer> map = new HashMap();
    HashMap<Integer, ListNode> nodeMap = new HashMap();
    LinkedListCustome list = new LinkedListCustome();
```



```java

    public FirstUnique(int[] nums) {
        for(int num : nums) {
            queue.add(num);
            //map.put(num, map.getOrDefault(num, 0) + 1);
            //if(map.get(num) == 1) {
            if(!nodeMap.containsKey(num)) {
                ListNode node = list.add(num);
                nodeMap.put(num, node);
            }
            else {
                ListNode node = nodeMap.get(num);
                if(node != null) {
                    list.remove(node);
                    //nodeMap.remove(num);
                    nodeMap.put(num, null);
                }
            }
        }
    }

    public int showFirstUnique() {
        ListNode head = list.head;

        if(head == null) {
            return -1;
        }
        else {
            return head.val;
        }
    }

    public void add(int value) {
        queue.add(value);
        //map.put(value, map.getOrDefault(value, 0) + 1);
        //if(map.get(value) == 1) {
        if(!nodeMap.containsKey(value)) {
            ListNode node = list.add(value);
            nodeMap.put(value, node);
        }
        else {
            ListNode node = nodeMap.get(value);
            if(node != null) {
                list.remove(node);
                //nodeMap.remove(value);
                nodeMap.put(value, null);
            }
        }
    }
}

/**
 * Your FirstUnique object will be instantiated and called as such:
 * FirstUnique obj = new FirstUnique(nums);
 * int param_1 = obj.showFirstUnique();
 * obj.add(value);
 */
```