Description | Editorial | Solutions (4.1K) | Submissions

414. Third Maximum Number

Easy 2.6K 2.9K

Companies

Given an integer array `nums`, return *the **third distinct maximum** number in this array. If the third maximum does not exist, return the **maximum** number.*

Example 1:

```
Input: nums = [3,2,1]
Output: 1
Explanation:
The first distinct maximum is 3.
The second distinct maximum is 2.
The third distinct maximum is 1.
```

Example 2:

```
Input: nums = [1,2]
Output: 2
Explanation:
The first distinct maximum is 2.
The second distinct maximum is 1.
The third distinct maximum does not exist, so the maximum (2) is
returned instead.
```

Example 3:

```
Input: nums = [2,2,3,1]
Output: 1
Explanation:
The first distinct maximum is 3.
The second distinct maximum is 2 (both 2's are counted together
```

Java | Auto

```java
class Solution {
    public int thirdMax(int[] nums) {
        int u =nums.length-1;
        int count = 0;

        for(int i=0;i<nums.length;i++){
            for(int j=1+i;j<nums.length;j++){
                if(nums[i] < nums[j]){
                    int temp = nums[i];
                    nums[i] = nums[j];
                    nums[j] = temp;
                }
            }
        }
        int max = nums[0];
        if(nums.length > 2){
            for(int i=0; i<nums.length; i++){
                if(nums[i] < max && count <2 && i != 0 && nums[i] != max){
                    max = nums[i];
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

nums =
[3,2,1]

Output

1

Expected

1

Console | Run | Submit

Description | Editorial | Solutions (5.2K) | Submissions

146. LRU Cache

Medium 18.2K 808

Companies

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]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation
```
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1);    // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2);    // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4,
```

Java | Auto

```java
class LRUCache {
    class Node {
        int key;
        int val;
        Node prev;
        Node next;

        Node(int key, int val) {
            this.key = key;
            this.val = val;
        }
    }

    Node head = new Node(-1, -1);
    Node tail = new Node(-1, -1);
    int cap;
    HashMap<Integer, Node> m = new HashMap<>();

    public LRUCache(int capacity) {
```

Testcase | Result

Accepted Runtime: 0 ms

Case 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]]

Output

[null,null,null,1,null,-1,null,-1,3,4]

Expected

Console | Run | Submit