716. Max Stack 2

Nov. 25, 2017 | 57.5K views

Average Rating: 3.80 (40 votes)

6 9 6

Сору

Сору

Next

Sort By ▼

Post

Design a max stack that supports push, pop, top, peekMax and popMax. 1. push(x) -- Push element x onto stack.

- 2. pop() -- Remove the element on top of the stack and return it.
- 3. top() -- Get the element on the top.
- 4. peekMax() -- Retrieve the maximum element in the stack.
- 5. popMax() -- Retrieve the maximum element in the stack, and remove it. If you find more than one
- maximum elements, only remove the top-most one. Example 1:

MaxStack stack = new MaxStack();

```
stack.push(5);
 stack.push(1);
 stack.push(5);
 stack.top(); -> 5
 stack.popMax(); -> 5
 stack.top(); -> 1
 stack.peekMax(); -> 5
 stack.pop(); -> 1
 stack.top(); -> 5
Note:
```

1. -1e7 <= x <= 1e7 Number of operations won't exceed 10000.

- The last four operations won't be called when stack is empty.

Approach #1: Two Stacks [Accepted]

For peekMax, we remember the largest value we've seen on the side. For example if we add [2, 1, 5, 3,

Intuition and Algorithm

9], we'll remember [2, 2, 5, 5, 9]. This works seamlessly with pop operations, and also it's easy to compute: it's just the maximum of the element we are adding and the previous maximum.

A regular stack already supports the first 3 operations, so we focus on the last two.

For popMax, we know what the current maximum (peekMax) is. We can pop until we find that maximum, then push the popped elements back on the stack. Our implementation in Python will showcase extending the list class.

Java Python 1 class MaxStack(list): def push(self, x):

```
m = max(x, self[-1][1] if self else None)
          self.append((x, m))
      def pop(self):
           return list.pop(self)[0]
        def top(self):
  9
  10
         return self[-1][0]
 11
 12
      def peekMax(self):
 13
         return self[-1][1]
 14
 15
       def popMax(self):
          m = self[-1][1]
 16
 17
           b = []
          while self[-1][0] != m:
 18
 19
              b.append(self.pop())
 20
 21
          self.pop()
 22
          map(self.push, reversed(b))
 23
            return m
Complexity Analysis
  ullet Time Complexity: O(N) for the popMax operation, and O(1) for the other operations, where N is
     the number of operations performed.
```

than O(N) time complexity.

map.get(x).add(node).

• Space Complexity: O(N), the maximum size of the stack.

largest value, insert values, and delete values, all in $O(\log N)$ time.

- Approach #2: Double Linked List + TreeMap [Accepted] Intuition
- Say we have a double linked list as our "stack". This reduces the problem to finding which node to remove, since we can remove nodes in O(1) time.

We can use a TreeMap mapping values to a list of nodes to answer this question. TreeMap can find the

Using structures like Array or Stack will never let us popMax quickly. We turn our attention to tree and

linked-list structures that have a lower time complexity for removal, with the aim of making popMax faster

Algorithm

Let's store the stack as a double linked list dll, and store a map from value to a List of Node.

• When we MaxStack.push(x), we add a node to our dll, and add or update our entry

When we MaxStack.pop(), we find the value val = dll.pop(), and remove the node from our

available.

Java

8

9 10

11 12

13

O Previous

Comments: 33

Preview

SHOW 2 REPLIES

SHOW 2 REPLIES

SHOW 3 REPLIES

SHOW 3 REPLIES

(1234)

1 A V C Share Reply

syed17 🛊 2 ② April 11, 2020 7:38 AM

2 A V C Share Share

You @awice for your contributions to the LC community.

Type comment here... (Markdown is supported)

1 class MaxStack {

}

public void push(int x) {

Node node = dll.add(x);

if(!map.containsKey(x))

map, deleting the entry if it was the last one.

value. The above operations are more clear given that we have a working DoubleLinkedList class. The

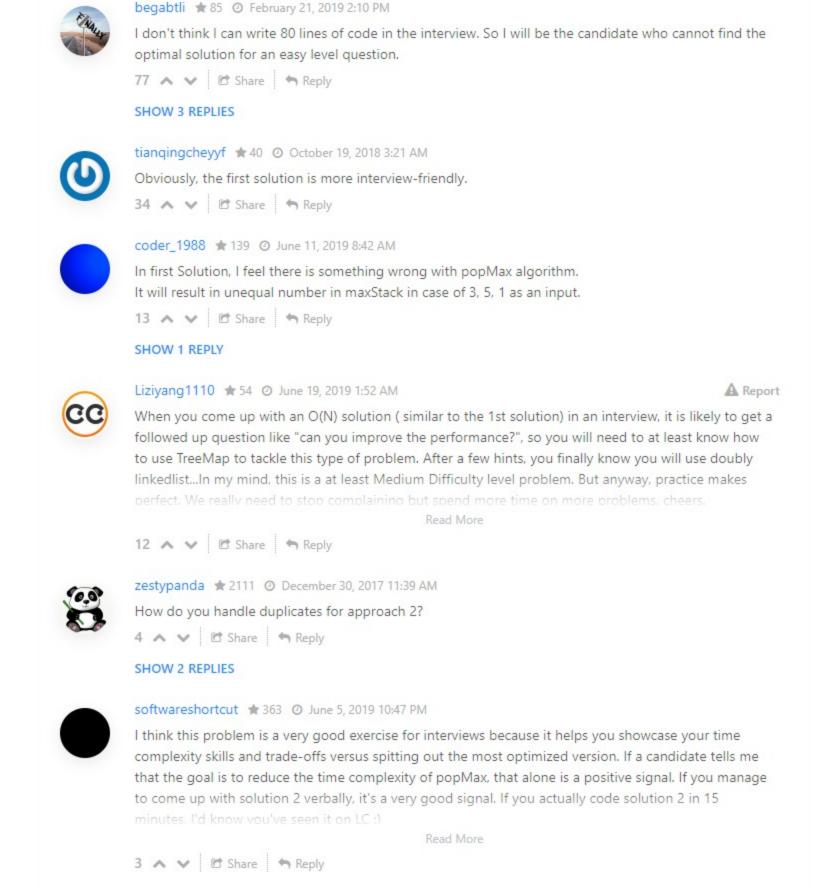
implementation provided uses head and tail sentinels to simplify the relevant DoubleLinkedList

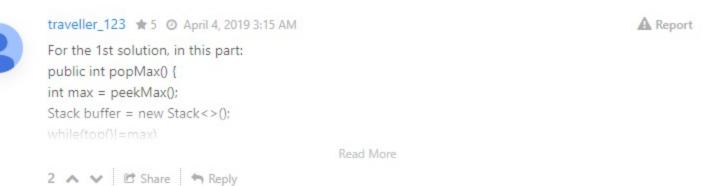
operations. A Python implementation was not included for this approach because there is no analog to *TreeMap*

When we MaxStack.popMax(), we use the map to find the relevant node to unlink, and return it's

TreeMap<Integer, List<Node>> map; DoubleLinkedList dll; 5 public MaxStack() { map = new TreeMap(); dll = new DoubleLinkedList();

```
map.put(x, new ArrayList<Node>());
 14
             map.get(x).add(node);
 15
         }
 16
         public int pop() {
 17
 18
            int val = dll.pop();
            List<Node> L = map.get(val);
 19
 20
             L.remove(L.size() - 1);
 21
            if (L.isEmpty()) map.remove(val);
 22
             return val;
 23
 24
 25
         public int top() {
 26
             return dll.peek();
 27
Complexity Analysis
   • Time Complexity: O(\log N) for all operations except peek which is O(1), where N is the number of
     operations performed. Most operations involving TreeMap are O(\log N).
   • Space Complexity: O(N), the size of the data structures used.
Analysis written by: @awice.
Rate this article: * * * * *
```





This should be a medium level question I believe, the explanation of the solution is very good. Thank

kevinhynes * 209 O December 12, 2019 8:55 PM A Report Approach #1 Python solution is broken. You cannot compare None with an int so the push method should be replaced with: def push(self, x):

Read More

dkasi 🛊 20 @ August 29, 2019 6:17 PM In the 1st Python solution, what happens in return list.pop(self)[0] ? Do you guys know why self is passed into pop? 1 A V C Share Reply