Max Stack

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
2. Number of operations won't exceed 10000.
3. The last four operations won't be called when stack is empty.

Solution 1

Because if it were, you could use this data structure to sort an array of numbers in O(n) time.

So, at the very least, either push(x) or popMax() must be O(logn)

written by brendon4565 original link here

Solution 2

The code is far from polished. The basic idea is to use BST.
1st treemap is to map from an always increasing index => real value. This can mimic a stack. I always get the last entry's value when pop.
2nd treemap is to map from real value => # of occurrences. This can help can get the max value easily.
I also maintain a hashmap that maps from real value => indices. When poping max, I get the last index of the value, and then remove that from the 1st treemap.
Each operation works in O(logn) time.

```
class MaxStack {

    TreeMap<Integer, Integer> tree; // index => value
    Map<Integer, List<Integer>> map; // value => indices
    TreeMap<Integer, Integer> max;
    int index = 0;

    /** initialize your data structure here. */
    public MaxStack() {
        tree = new TreeMap<>();
        map = new HashMap<>();
        max = new TreeMap<>();
    }

    public void push(int x) {
        index++;
        tree.put(index, x);
        map.computeIfAbsent(x, e -> new ArrayList<>()).add(index);
        max.put(x, max.getOrDefault(x, 0)+1);
    }

    public int pop() {
        int k = tree.lastKey();
        int ret = tree.remove(k);
        map.get(ret).remove(map.get(ret).size()-1);
        if (map.get(ret).isEmpty()) map.remove(ret);
        max.put(ret, max.get(ret)-1);
        if (max.get(ret) == 0) max.remove(ret);

        return ret;
    }

    public int top() {
        return tree.lastEntry().getValue();
    }

    public int peekMax() {
        return max.lastKey();
    }

    public int popMax() {
        int ret = max.lastKey();
        max.put(ret, max.get(ret)-1);
        if (max.get(ret) == 0) max.remove(ret);
```

```
        int idx = map.get(ret).get(map.get(ret).size()-1);
        map.get(ret).remove(map.get(ret).size()-1);
        if (map.get(ret).isEmpty()) map.remove(ret);

        tree.remove(idx);

        return ret;
    }
}

/**
 * Your MaxStack object will be instantiated and called as such:
 * MaxStack obj = new MaxStack();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.peekMax();
 * int param_5 = obj.popMax();
 */
```

written by azrush original link here

Solution 3

Passed all the test cases, but not sure whether it covers all.

s1 stores all the values, s2 stores the max values, and s3 stores the values popped from s1 during the popMax() steps.

```java
class MaxStack {
    Stack<Integer> s1, s2, s3;
    /** initialize your data structure here. */
    public MaxStack() {
        s1 = new Stack<>();
        s2 = new Stack<>();
        s3 = new Stack<>();
    }

    public void push(int x) {
        s1.push(x);
        if (s2.isEmpty() || x >= s2.peek())
            s2.push(x);
    }

    public int pop() {
        int val = s1.pop();
        if (!s2.isEmpty() && s2.peek() == val)
            s2.pop();
        return val;
    }

    public int top() {
        return s1.peek();
    }

    public int peekMax() {
        if (s2.isEmpty())
            return s1.peek();
        return s2.peek();
    }

    public int popMax() {
        int max = !s2.isEmpty() ? s2.pop() : s1.peek();
        while (!s1.isEmpty() && s1.peek() != max)
            s3.push(s1.pop());

        s1.pop();
        while (!s3.isEmpty())
            push(s3.pop());
        return max;
    }
}

/**
 * Your MaxStack object will be instantiated and called as such:
 * MaxStack obj = new MaxStack();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.peekMax();
 * int param_5 = obj.popMax();
 */
```

written by llian original link here

From Leetcoder.