155.最小栈

亚马逊在半年内面试常考

最小栈

155. 最小栈

难度 简单 957 收藏 分享 切换为英文 接收动态 反馈

设计一个支持 `push` ，`pop` ，`top` 操作，并能在常数时间内检索到最小元素的栈。

- `push(x)` —— 将元素 x 推入栈中。
- `pop()` —— 删除栈顶的元素。
- `top()` —— 获取栈顶元素。
- `getMin()` —— 检索栈中的最小元素。

示例:

```
输入:
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

输出:
[null,null,null,null,-3,null,0,-2]

解释:
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.getMin();   --> 返回 -2.
```

提示:

- `pop`、`top` 和 `getMin` 操作总是在 **非空栈** 上调用。

155. Min Stack

难度 简单 957 收藏 分享 切换为中文 接收动态 反馈

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

Example 1:

```
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top();    // return 0
minStack.getMin(); // return -2
```

Constraints:

- $-2^{31} <= val <= 2^{31} - 1$
- Methods `pop`, `top` and `getMin` operations will always be called on **non-empty** stacks.
- At most $3 * 10^4$ calls will be made to `push`, `pop`, `top`, and `getMin`.

思路 1 ： 辅助栈

1. 初始化两个栈：
   - normal_stack : 正常栈，控制压栈和出栈；
   - min_stack : 最小值辅助栈，来存取、获取 normal_stack 中的最小值.
2. push() 方法：
   - 每当 push() 新值进来时，normal_stack 压入新值；
   - **更新最小值** ：如果新值小于等于 min_stack 栈顶值，则 min_stack 压入新值，
3. pop() 方法：
   - normal_stack 栈 pop 出栈顶值；
   - **最小值同步** ： 判断从 normal_stack 中 pop 出的值是否是 min_stack 栈顶元素（即最小值），如果是，则 min_stack 也 pop 出栈顶值，来保证 min_stack 栈顶值始终是 normal_stack 的最小值.

代码：

```
// Java
// Time : 2021 - 07 - 19

class MinStack {

    private Stack<Integer> normal_stack;    // control push、pop operation
    private Stack<Integer> min_stack;       // record minmum value in stack

    public MinStack() {

        // initialization two stacks
        normal_stack = new Stack<>();
        min_stack = new Stack<>();
    }

    // push operation
    public void push(int x) {

        normal_stack.push(x);   // push element in to normal_stack

        // min_stack is empty && pushing value <= top value in min_stack
        // push elments into min_stack
        if (min_stack.isEmpty() || x <= min_stack.peek())
            min_stack.push(x);

        // note : min_stack.peek() > x -> error
    }

    // pop operation
    public void pop() {
        // pop top value from normal_stack
        // and compare whether equals to top value in min_stack
        // if euqals, pop min_stack top value to keep the top value is always
        // the minmum value for normal_stack
        if (normal_stack.pop().equals(min_stack.peek()))
```

```
            min_stack.pop();

        // equals() method is different to == operator
        // equals() : evaluates to the comparison of values in the objects
        // == : checks if both objects point to the same memory location
        // if use == operator -> error
    }

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

    public int getMin() {
        return min_stack.peek();
    }
}
```

复杂度分析：

- 时间复杂度 O(1)：压栈，出栈，获取最小值的时间复杂度都为 O(1).
- 空间复杂度 O(n)： N 个元素辅助栈会占用线性大小的额外空间。

#Leetcode/Stack#