# 题目

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。

**注：**

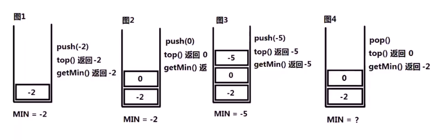
面试题30

# 分析

## 方法一：辅助栈

可以定义一个变量minEle存储最小值，在入栈的时候依次将入栈元素与之比较，小于该值则更新该值为入栈元素，否则继续。在获取最小元素的时候，直接返回该变量即可。但是，这种方案在pop操作的时候存在问题：比如push元素为3,2,1，minEle=1，pop出1的时候，该值应该更新为2，而不应该为1。也就是说，我们在push操作更新minEle的时候不应该将之前的旧的最小值覆盖掉，应该保存起来pop的时候使用。

采用1个变量记录最小值：



结论：

1、1个变量Min无法完成记录栈中所有状态下的最小值（pop操作被覆盖）；

2、栈的每个状态，都需要有一个变量记录最小值

应该使用数组依次记录入栈时最小元素，执行push操作时（入栈顺序3->2->1->2->3）：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 2 | 1 | 2 | 3 |

入栈操作对应的最小元素数组为：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 2 | 1 | 1 | 1 |

执行pop操作的时候，两者的变化为：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 2 | 1 | 2 |  |
| 3 | 2 | 1 | 1 |  |

继续：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 2 | 1 |  |  |
| 3 | 2 | 1 |  |  |

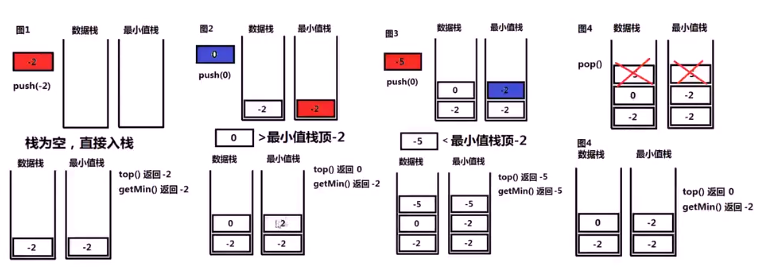
继续：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 2 |  |  |  |
| 3 | 2 |  |  |  |

继续：

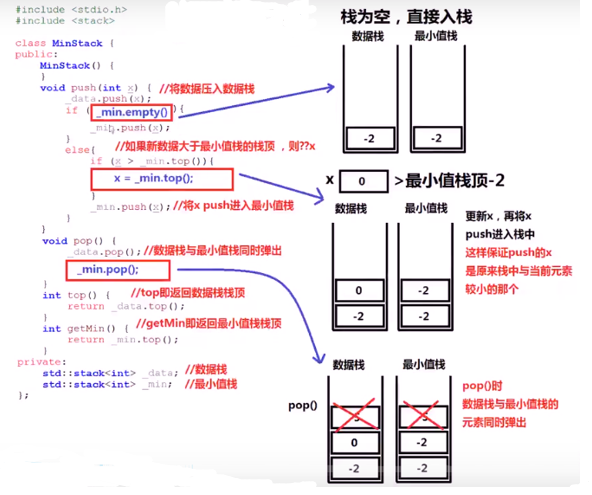
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 |  |  |  |  |
| 3 |  |  |  |  |

图示：



注：剑指offer P132（非常重要！） Leetcode155

**代码：**



class MinStack {

private:

    stack<int> \_data;

    stack<int> \_min;

public:

    /\*\* initialize your data structure here. \*/

    MinStack() {

    }

    void push(int x) {

        \_data.push(x);

//更新\_data栈后更新存储当前最小值的栈

        if (\_min.empty()){

            \_min.push(x);

        }else{

            if(x > \_min.top()){

                x = \_min.top();

            }

            \_min.push(x);

        }

    }

    void pop() {

        \_data.pop();

        \_min.pop();

    }

    int top() {

        return \_data.top();

    }

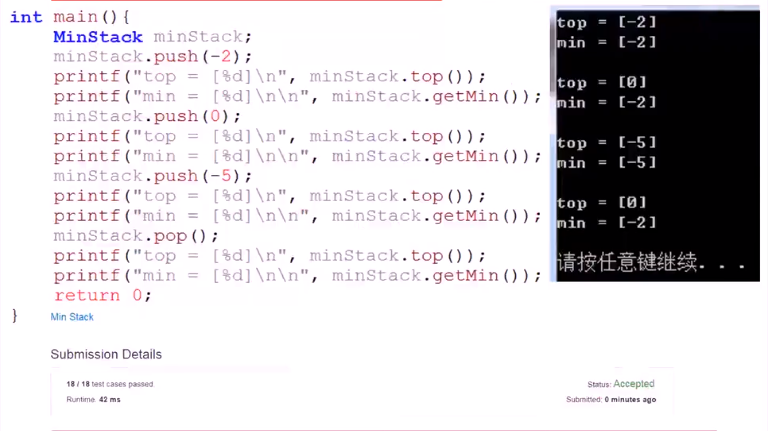
    int min() {

        return \_min.top();

    }

};

**测试：**



另一种写法：

class MinStack {

private:

    stack<int> \_data;

    stack<int> \_min;

public:

    MinStack() {

    }

    void push(int val) {

        \_data.push(val);

        if(\_min.empty())

        {

            \_min.push(val);

        }

        else

        {

            if(val < \_min.top())

            {

                \_min.push(val);

            }

            else

            {

                \_min.push(\_min.top());

            }

        }

    }

    void pop() {

        \_data.pop();

        \_min.pop();

    }

    int top() {

        return \_data.top();

    }

    int getMin() {

        return \_min.top();

    }

};