155 Min Stack

2018年4月3日 13:49

Question:

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

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

Example:

MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> Returns -3.
minStack.pop();
minStack.top();   --> Returns 0.
minStack.getMin();   --> Returns -2.

来自 <https://leetcode.com/problems/min-stack/description/>

设计一个支持 push，pop，top 操作，并能在常量时间内检索最小元素的栈。

- 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。

Solution for Python3:

```
from collections import deque
class MinStack1:
    def __init__(self):
        """
        initialize your data structure here.
        """
        self.d1 = deque()
        self.d2 = deque()


    def push(self, x):
        """
        :type x: int
        :rtype: void
       """
        self.d1.append(x)
        if (not self.d2 or x <= self.getMin()):
           self.d2.append(x)

```

```python
    def pop(self):
        """
        :rtype: void
        """
        if self.d1[-1] == self.getMin():
            self.d2.pop()
        self.d1.pop()

    def top(self):
        """
        :rtype: int
        """
        return self.d1[-1]


    def getMin(self):
        """
        :rtype: int
        """
        return self.d2[-1]


class MinStack2:

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.L = []


    def push(self, x):
        """
        :type x: int
        :rtype: void
        """
        self.L.append((x, min(self.getMin(), x)))


    def pop(self):
        """
        :rtype: void
        """
        self.L.pop()


    def top(self):
        """
        :rtype: int
        """
        if self.L:
            return self.L[-1][0]
        return None


    def getMin(self):
        """
        :rtype: int
        """
        if self.L:
            return self.L[-1][1]
        return sys.maxsize


# Your MinStack object will be instantiated and called as such:
# obj = MinStack()
# obj.push(x)
# obj.pop()
# param_3 = obj.top()
# param_4 = obj.getMin()
```

Solution for C++:

```
class MinStack1 {
private:
    stack<int> s1, s2;
public:
    /** initialize your data structure here. */
    MinStack() {

    }

    void push(int x) {
        s1.push(x);
        if (s2.empty() || x <= getMin()) {
            s2.push(x);
        }
    }

    void pop() {
        if (s1.top() == getMin) {
            s2.pop();
        }
        s1.pop();
    }

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

    int getMin() {
        return s2.top();
    }
};


class MinStack2 {
private:
    vector<pair<int, int>> v;
public:
    /** initialize your data structure here. */
    MinStack() {

    }

    void push(int x) {
       v.push_back(make_pair(x, min(x, getMin())));
    }

    void pop() {
       v.pop_back();
    }

    int top() {
       if (!v.empty())
           return v.back().first;
           // return (*(v.end() - 1)).first;
        return NULL;
    }

    int getMin() {
       if (!v.empty())
           return v.back().second;
           // return (*(v.end() - 1)).second;
        return INT_MAX;
    }
};


/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack obj = new MinStack();
 * obj.push(x);
 * obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.getMin();
 */
```