232 Implement Queue using Stacks

2018年4月5日 17:16

Question:

Implement the following operations of a queue using stacks.

- push(x) -- Push element x to the back of queue.
- pop() -- Removes the element from in front of queue.
- peek() -- Get the front element.
- empty() -- Return whether the queue is empty.

Notes:

- You must use *only* standard operations of a stack -- which means only push to top, peek/pop from top, size, and is empty operations are valid.
- Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.
- You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

来自 <https://leetcode.com/problems/implement-queue-using-stacks/description/>

使用栈来实现队列的如下操作：

- push(x) -- 将一个元素放入队列的尾部。
- pop() -- 从队列首部移除元素。
- peek() -- 返回队列首部的元素。
- empty() -- 返回队列是否为空。

注意：

- 你只能使用标准的栈操作-- 也就是只有push to top, peek/pop from top, size, 和 is empty 操作是可使用的。
- 你所使用的语言也许不支持栈。你可以使用 list 或者 deque（双端队列）来模拟一个栈，只要你仅使用栈的标准操作就可以。
- 假设所有操作都是有效的，比如 pop 或者 peek 操作不会作用于一个空队列上。

Solution for Python3:

```python
class MyQueue:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        from collections import deque
        self.input, self.output = deque(), deque()


    def push(self, x):
        """
        Push element x to the back of queue.
        :type x: int
        :rtype: void
        """
        self.input.append(x)


    def pop(self):
        """
        Removes the element from in front of queue and returns that element.
        :rtype: int
```

```
        """
        self.peek()
        return self.output.pop()


    def peek(self):
        """
        Get the front element.
        :rtype: int
        """
        if not self.output:
           while self.input:
               self.output.append(self.input.pop())
        return self.output[-1]


    def empty(self):
        """
        Returns whether the queue is empty.
        :rtype: bool
        """
        return not len(self.input) and not len(self.output)


# Your MyQueue object will be instantiated and called as such:
# obj = MyQueue()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.peek()
# param_4 = obj.empty()
```

Solution for C++:

```
class MyQueue {
public:
    /** Initialize your data structure here. */
    stack<int> input, output;
    MyQueue() {

    }

    /** Push element x to the back of queue. */
    void push(int x) {
        input.push(x);
    }

    /** Removes the element from in front of queue and returns that element. */
    int pop() {
        int t = peek();
        output.pop();
        return t;
    }

    /** Get the front element. */
    int peek() {
        if (output.empty()) {
```

```
            while (!input.empty()) {
                output.push(input.top());
                input.pop();
            }
        }
        return output.top();
    }

    /** Returns whether the queue is empty. */
    bool empty() {
        return input.empty() and output.empty();
    }
};
```