均摊复杂度O(1)

2022年8月15日 8:27

原题：leetcode232 用栈实现队列

232. 用栈实现队列

难度 简单 729 收藏 分享 切换为英文 接收动态 反馈

请你仅使用两个栈实现先入先出队列。队列应当支持一般队列支持的所有操作（`push`、`pop`、`peek`、`empty`）：

实现 `MyQueue` 类：

- `void push(int x)` 将元素 x 推到队列的末尾
- `int pop()` 从队列的开头移除并返回元素
- `int peek()` 返回队列开头的元素
- `boolean empty()` 如果队列为空，返回 `true`；否则，返回 `false`

说明：

- 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` 操作是合法的。
- 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，只要是标准的栈操作即可。

所实现的pop()方法以及peek()方法 其复杂度 均为均摊O(1).

```
void in2out(){
    while(!s_in.empty()){
        s_out.push(s_in.top());
        s_in.pop();
    }
}
void push(int x) {
    s_in.push(x);
}

int pop() {
    while(s_out.empty()){
        this->in2out();
    }
    int temp=s_out.top();
    s_out.pop();
    return temp;
}

int peek() {
    while(s_out.empty()){
        this->in2out();
    }
    return s_out.top();
}
```

我们先用另外一个例子来理解「均摊复杂度」，大家都知道「哈希表」底层是通过数组实现的。

正常情况下，计算元素在哈希桶的位置，然后放入哈希桶，复杂度为 $O(1)$，假定是通过简单的“拉链法”搭配「头插法」方式来解决哈希冲突。

但当某次元素插入后，「哈希表」达到扩容阈值，则需要对底层所使用的数组进行扩容，这个复杂度是 $O(n)$

显然「扩容」操作不会发生在每一次的元素插入中，因此扩容的 $O(n)$ 都会伴随着 `n` 次的 $O(1)$，也就是 $O(n)$ 的复杂度会被均摊到每一次插入当中，因此哈希表插入仍然是 $O(1)$ 的。

同理，我们的「倒腾」不是发生在每一次的「输出操作」中，而是集中发生在一次「输出栈为空」的时候，因此 `pop` 和 `peek` 都是均摊复杂度为 $O(1)$ 的操作。