

数据结构设计类问题

主讲人 令狐冲



最小栈

https://www.lintcode.com/problem/min-stack/

设计一个栈,支持 O(1) push, pop, min 和堆的区别是啥?



```
public class MinStack {
   Stack<Integer> stack, minStack;
   public MinStack() {
       stack = new Stack<Integer>();
       minStack = new Stack<Integer>();
   public void push(int number) {
       stack.push(number);
       if (minStack.empty() || number < minStack.peek()) {</pre>
           minStack.push(number);
       } else {
           minStack.push(minStack.peek());
   public int pop() {
       minStack.pop();
       return stack.pop();
   public int min() {
       return minStack.peek();
```

```
class MinStack:
   def __init__(self):
       self.stack = []
       self.min_stack = []
   def push(self, number):
       self.stack.append(number)
       if self.min_stack and number > self.min_stack[-1]:
            self.min_stack.append(self.min_stack[-1])
       else:
            self.min_stack.append(number)
   def pop(self):
       self.min_stack.pop()
       return self.stack.pop()
   def min(self):
       return self.min_stack[-1]
```



有什么可以优化的地方么?

空间方面



```
public class MinStack {
   Stack<Integer> stack, minStack;
   public MinStack() {
       stack = new Stack<Integer>();
       minStack = new Stack<Integer>();
   public void push(int number) {
       stack.push(number);
       if (minStack.empty() || number <= minStack.peek()) {</pre>
           minStack.push(number);
   public int pop() {
       int number = stack.pop();
       if (number == minStack.peek()) {
           minStack.pop();
       return number;
   public int min() {
       return minStack.peek();
```

```
class MinStack:
    def init (self):
       self.stack = []
        self.min_stack = []
    def push(self, number):
       self.stack.append(number)
       if not self.min_stack or self.min_stack[-1] >= number:
            self.min_stack.append(number)
    def pop(self):
       number = self.stack.pop()
       if number == self.min_stack[-1]:
            self.min_stack.pop()
        return number
    def min(self):
        return self.min_stack[-1]
```



最大栈

https://www.lintcode.com/problem/max-stack/

实现 push, pop, top, peekMax, popMax

Java 代码



```
class MaxStack {
    Stack<Integer> stack;
    Stack<Integer> maxStack;
    public MaxStack() {
        stack = new Stack();
        maxStack = new Stack();
    public void push(int x) {
        int max = maxStack.isEmpty() ? x : maxStack.peek();
        maxStack.push(max > x ? max : x);
        stack.push(x);
    public int pop() {
        maxStack.pop();
        return stack.pop();
```

```
public int top() {
    return stack.peek();
public int peekMax() {
    return maxStack.peek();
public int popMax() {
    int max = peekMax();
    Stack<Integer> buffer = new Stack();
    while (top() != max) {
        buffer.push(pop());
    pop();
    while (!buffer.isEmpty()) {
        push(buffer.pop());
    return max;
```

Python 代码



```
class MaxStack:
    def __init__(self):
        self.stack = []
        self.max_stack = []
    def push(self, x):
        self.stack.append(x)
        if self.max_stack:
            number = max(self.max_stack[-1], x)
            self.max_stack.append(number)
        else:
            self.max_stack.append(x)
    def pop(self):
        self.max_stack.pop()
        return self.stack.pop()
```

```
def top(self):
    return self.stack[-1]
def peekMax(self):
    return self.max_stack[-1]
    popMax(self):
    max_number = self.peekMax()
    buffer_stack = []
    while self.top() != max_number:
        buffer_stack.append(self.pop())
    self.pop()
    while buffer_stack:
        self.push(buffer_stack[-1])
        buffer_stack.pop()
    return max_number
```



一个更快的办法

使用 heap + stack + hashset 结合的办法
用 hashset 的作用是标记被 pop 和 popMax 的那些数
用 heap 的作用是为了快速找到大的数
用 stack 的作用是为了找到位置上最大的数
如何处理重复的数?

Python 代码



```
import heapq
class MaxStack:
   def __init__(self):
       self.heap = []
       self.stack = []
       self.popped_set = set()
       self.count = 0
   def push(self, x):
       item = (-x, -self.count)
       self.stack.append(item)
       heapq.heappush(self.heap, item)
       self.count += 1
   def _clear_popped_in_stack(self):
       while self.stack and self.stack[-1] in self.popped_set:
           self.popped_set.remove(self.stack[-1])
           self.stack.pop()
   def _clear_popped_in_heap(self):
       while self.heap and self.heap[0] in self.popped_set:
           self.popped_set.remove(self.heap[0])
           heapq.heappop(self.heap)
```

```
def pop(self):
    self._clear_popped_in_stack()
    item = self.stack.pop()
    self.popped_set.add(item)
    return -item[0]
def top(self):
    self._clear_popped_in_stack()
    item = self.stack[-1]
    return -item[0]
def peekMax(self):
    self._clear_popped_in_heap()
    item = self.heap[0]
    return -item[0]
def popMax(self):
    self._clear_popped_in_heap()
    item = heapq.heappop(self.heap)
    self.popped_set.add(item)
    return -item[0]
```

Java 代码



```
lass Item implements Comparable<Item> {
   public int val, id;
   public Item(int val, int id) {
       this.val = val;
       this.id = id;
   public int compareTo(Item another) {
       if (this.val != another.val) {
           return another.val - this.val;
       return another.id - this.id;
class MaxStack {
   private Queue<Item> heap;
  private Stack<Item> stack;
   private HashSet<Item> poppedSet;
   private int globalId;
   public MaxStack() {
       this.globalId = 0;
       this.heap = new PriorityQueue<>();
       this.stack = new Stack<>();
       this.poppedSet = new HashSet<>();
```

```
public void push(int x) {
    Item item = new Item(x, globalId);
    stack.push(item);
    heap.offer(item);
    globalId++;
}

private void clearPoppedInStack() {
    while (!stack.isEmpty() && poppedSet.contains(stack.peek())) {
        Item item = stack.pop();
        poppedSet.remove(item);
    }
}

private void clearPoppedInHeap() {
    while (!heap.isEmpty() && poppedSet.contains(heap.peek())) {
        Item item = heap.poll();
        poppedSet.remove(item);
    }
}
```

```
public int pop() {
    clearPoppedInStack();
    Item item = stack.pop();
    poppedSet.add(item);
    return item.val;
public int top() {
    clearPoppedInStack();
    Item item = stack.peek();
    return item.val;
public int peekMax() {
    clearPoppedInHeap();
    Item item = heap.peek();
    return item.val;
public int popMax() {
    clearPoppedInHeap();
    Item item = heap.poll();
    poppedSet.add(item);
    return item.val;
```



时间复杂度是多少?

数据结构设计类问题需要分别说明不同函数的时间复杂度

时间复杂度分析



因为每个数只进入 stack/heap/hashset 各一次。所以平均下来,时间复杂度是:

- push O(logN)
- pop O(1)
- top O(1)
- popMax O(logN)
- peekMax O(logN)



两个栈实现队列

https://www.lintcode.com/problem/implement-queue-by-two-stacks/除了栈以外不能使用其他数据结构



```
oublic class MyQueue {
  private Stack<Integer> stack1;
  private Stack<Integer> stack2;
  public MyQueue() {
     stack1 = new Stack<Integer>();
     stack2 = new Stack<Integer>();
  private void stack2ToStack1(){
      while(! stack2.isEmpty()){
           stack1.push(stack2.pop());
  public void push(int element) {
      stack2.push(element);
  public int pop() {
       if(stack1.empty() == true){
          this.stack2ToStack1();
       return stack1.pop();
  public int top() {
       if(stack1.empty() == true){
           this.stack2ToStack1();
       return stack1.peek();
```

```
class MyQueue:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []
    def stack1_to_stack2(self):
        # 如果 stack2 非空的话, 不把 stack1 倒过去
        if self.stack2:
            return
        while self.stack1:
            self.stack2.append(self.stack1.pop())
    def push(self, element):
        self.stack1.append(element)
    def top(self):
        self.stack1_to_stack2()
        return self.stack2[-1]
    def pop(self):
        self.stack1_to_stack2()
        return self.stack2.pop()
```



时间复杂度是多少?

数据结构设计类问题需要分别说明不同函数的时间复杂度



两个队列实现栈

https://www.lintcode.com/problem/implement-stack-by-two-queues/

只能使用队列,不能使用其他数据结构

Java 代码



```
public class Stack {
   private Queue<Integer> queue1;
   private Queue<Integer> queue2;
   public Stack() {
       queue1 = new LinkedList<Integer>();
       queue2 = new LinkedList<Integer>();
   private void moveItems() {
       while (queue1.size() != 1) {
           queue2.offer(queue1.poll());
   private void swapQueues() {
       Queue<Integer> temp = queue1;
       queue1 = queue2;
       queue2 = temp;
    * push a new item into the stack
   public void push(int value) {
       queue1.offer(value);
```

```
* return the top of the stack
public int top() {
    int item = pop();
    queue1.offer(item);
   return item;
* pop the top of the stack and return it
public int pop() {
   moveItems();
    int item = queue1.poll();
   swapQueues();
    return item;
* check the stack is empty or not.
public boolean isEmpty() {
    return queue1.isEmpty();
```

Python 代码



```
class Stack:
    def __init__(self):
        self.queue1 = deque()
        self.queue2 = deque()
    11 11 11
    @param: x: An integer
    @return: nothing
    def push(self, x):
        self.queue1.append(x)
    11 11 11
    @return: nothing
       pop(self):
        for _ in range(len(self.queue1) - 1):
            val = self.queue1.popleft()
            self.queue2.append(val)
        val = self.queue1.popleft()
        self.queue1, self.queue2 = self.queue2, self.queue1
        return val
```

```
11 11 11
@return: An integer
def top(self):
    val = self.pop()
    self.push(val)
    return val
11 11 11
@return: True if the stack is empty
11 11 11
def isEmpty(self):
    return not self.queue1
```



时间复杂度是多少?

数据结构设计类问题需要分别说明不同函数的时间复杂度