题目地址(1381. 设计一个支持增量操作的栈)

https://leetcode-cn.com/problems/design-a-stack-with-increment-operation/

入选理由

1. 前两天是数组，今后三天都是栈。 栈的难度不低，值得大家注意。队列的应用我们放到后面 bfs 和 堆部分出题

2. 难度中等。可以和前两天的题目形成难度梯度。

题目描述

请你设计一个支持下述操作的栈。

实现自定义栈类 CustomStack ：

CustomStack(int maxSize)：用 maxSize 初始化对象，maxSize 是栈中最多能容纳的元素数量，栈在增长到 maxSize 之后则不支持 push 操作。
void push(int x)：如果栈还未增长到 maxSize ，就将 x 添加到栈顶。
int pop()：弹出栈顶元素，并返回栈顶的值，或栈为空时返回 -1 。
void inc(int k, int val)：栈底的 k 个元素的值都增加 val 。如果栈中元素总数小于 k ，则栈中的所有元素都增加 val 。

示例：

输入：
["CustomStack","push","push","pop","push","push","push","increment","increment","pop","pop","pop","pop"]
[[3],[1],[2],[],[2],[3],[4],[5,100],[2,100],[],[],[],[]]
输出：
[null,null,null,2,null,null,null,null,null,103,202,201,-1]
解释：

```
CustomStack customStack = new CustomStack(3); // 栈是空的 []
customStack.push(1); // 栈变为 [1]
customStack.push(2); // 栈变为 [1, 2]
customStack.pop(); // 返回 2 --> 返回栈顶值 2，栈变为 [1]
customStack.push(2); // 栈变为 [1, 2]
customStack.push(3); // 栈变为 [1, 2, 3]
customStack.push(4); // 栈仍然是 [1, 2, 3]，不能添加其他元素使栈大小变为 4
customStack.increment(5, 100); // 栈变为 [101, 102, 103]
customStack.increment(2, 100); // 栈变为 [201, 202, 103]
customStack.pop(); // 返回 103 --> 返回栈顶值 103，栈变为 [201, 202]
customStack.pop(); // 返回 202 --> 返回栈顶值 202，栈变为 [201]
customStack.pop(); // 返回 201 --> 返回栈顶值 201，栈变为 []
customStack.pop(); // 返回 -1 --> 栈为空，返回 -1
```

提示：

1 <= maxSize <= 1000

```
1 <= x <= 1000
1 <= k <= 1000
0 <= val <= 100
每种方法 increment，push 以及 pop 分别最多调用 1000 次
```

难度

- 简单

标签

- 栈

前置知识

- 栈
- 前缀和

increment 时间复杂度为 $O(k)$ 的方法

思路

首先我们来看一种非常符合直觉的方法，然而这种方法并不好，increment 操作需要的时间复杂度为 $O(k)$。

`push` 和 `pop` 就是普通的栈操作。 唯一要注意的是边界条件，这个已经在题目中指明了，具体来说就是：

- push 的时候要判断是否满了
- pop 的时候要判断是否空了

而做到上面两点，只需要一个 cnt 变量记录栈的当前长度，一个 size 变量记录最大容量，并在 pop 和 push 的时候更新 cnt 即可。

代码

代码支持：Python，JS，Java，CPP

Python Code：

```python
class CustomStack:

    def __init__(self, size: int):
        self.st = []
```

```
        self.cnt = 0
        self.size = size

    def push(self, x: int) -> None:
        if self.cnt < self.size:
            self.st.append(x)
            self.cnt += 1


    def pop(self) -> int:
        if self.cnt == 0: return -1
        self.cnt -= 1
        return self.st.pop()


    def increment(self, k: int, val: int) -> None:
        for i in range(0, min(self.cnt, k)):
            self.st[i] += val
```

JS Code:

```
/**
 * @param {number} maxSize
 */
var CustomStack = function (maxSize) {
  this.max = maxSize;
  this.stack = [];
};

/**
 * @param {number} x
 * @return {void}
 */
CustomStack.prototype.push = function (x) {
  if (this.stack.length < this.max) {
    this.stack.push(x);
  }
};

/**
 * @return {number}
 */
CustomStack.prototype.pop = function () {
  var res = this.stack.pop();
  return res == null ? -1 : res;
};

/**
```

```
 * @param {number} k
 * @param {number} val
 * @return {void}
 */
CustomStack.prototype.increment = function (k, val) {
  for (var i = 0; i < this.stack.length; i++) {
    if (i < k) {
      this.stack[i] += val;
    }
  }
};
```

Java Code:

```
public class CustomStack {
    int[] stack;
    int top;
    public CustomStack(int maxSize) {
        stack = new int[maxSize];
        top =-1;
    }
    public void Push(int x) {
        if(top!=stack.Length-1)
        {
            top++;
            stack[top]=x;
        }
    }
    public int Pop() {
        if(top==-1)
        {
            return -1;
        }
        --top;
        return stack[top + 1];
    }
    public void Increment(int k, int val) {
        int limit = Math.Min(k, top + 1);
        for (int i = 0; i < limit; ++i)
        {
            stack[i] += val;
        }
    }
}
```

CPP Code:

```
class CustomStack {
    maxSize:number;
    cnt:number;
    stack: Array<number>;
    incrementInfos: Array<number>;

    constructor(maxSize: number) {
      this.maxSize = maxSize;
      this.cnt = -1;
      this.incrementInfos = new Array(maxSize).fill(0);
      this.stack = [];
    }

    push(x: number): void {
      if (this.cnt < this.maxSize -1) {
        this.cnt++;
        this.stack.push(x);
      }
    }

    pop(): number {
      if (this.cnt === -1) return -1;
      const inc = this.incrementInfos[this.cnt];

      if (inc) {
        this.incrementInfos[this.cnt] = 0;
        this.incrementInfos[this.cnt -1] += inc;
      }
      this.cnt--;
      return this.stack.pop() + inc;
    }

    increment(k: number, val: number): void {
      let i = k;
      if (this.cnt < i) i = this.cnt + 1;
      if (i > 0 ) this.incrementInfos[i - 1] += val;
    }
}
```

复杂度分析

- 时间复杂度：push 和 pop 操作的时间复杂度为 $O(1)$（讲义有提到），而 increment 操作的时间复杂度为 $O(\min(k, cnt))$
- 空间复杂度：$O(1)$

前缀和

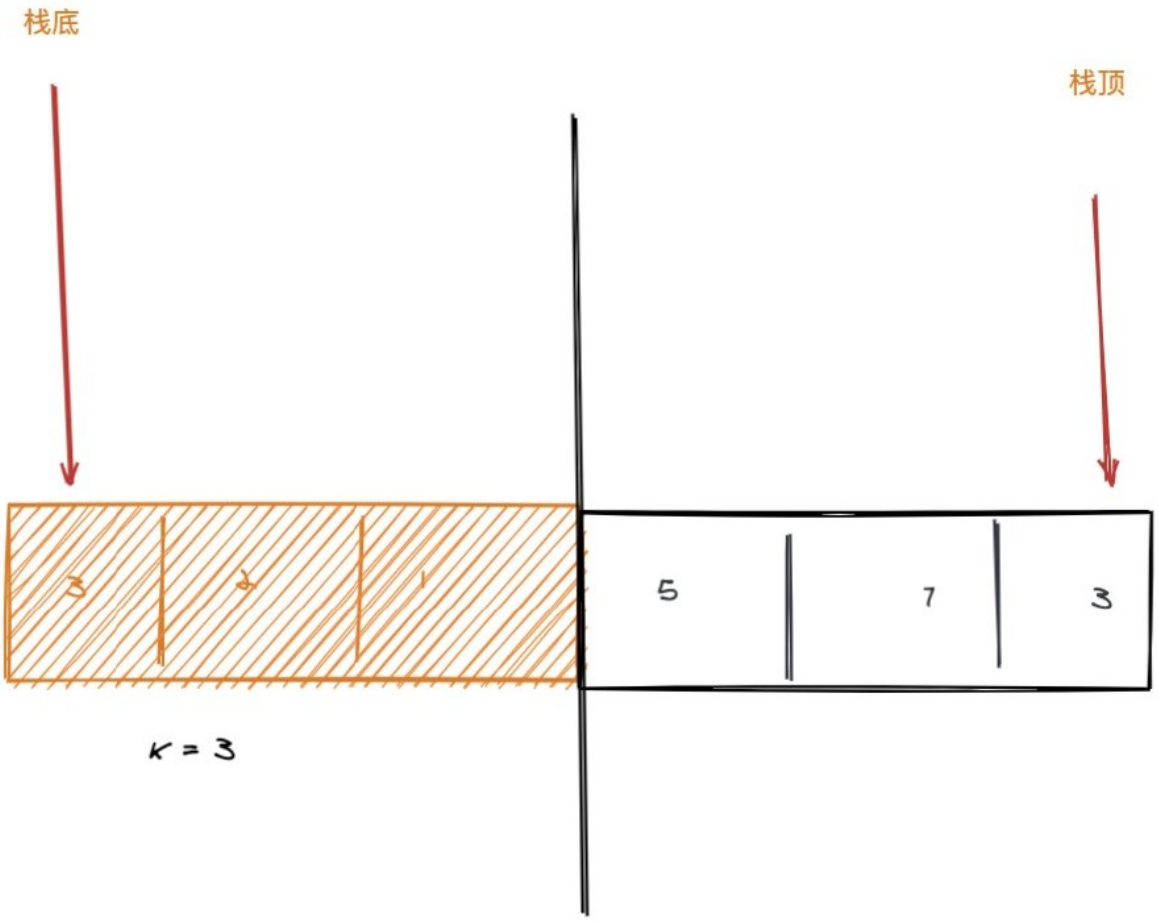
前缀和在讲义里面提到过，大家也可是看下我的文章 [一次搞定前缀和]

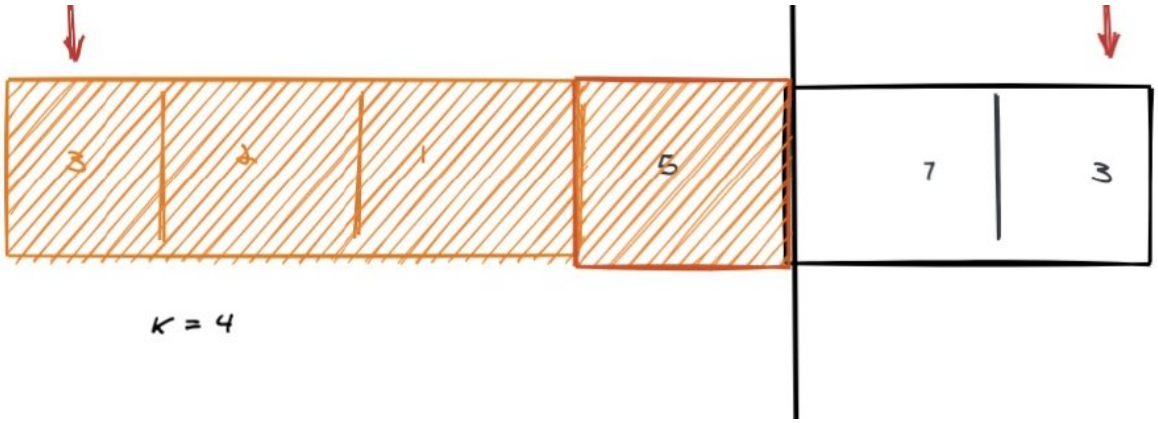
思路

和上面的思路类似，不过我们采用空间换时间的方式。采用一个额外的数组 incrementals 来记录每次 incremental 操作。

具体算法如下：

- 初始化一个大小为 maxSize 的数组 incrementals， 并全部填充 0
- push 操作不变，和上面一样
- increment 的时候，我们将用到 incremental 信息。那么这个信息是什么，从哪来呢？我这里画了一个图

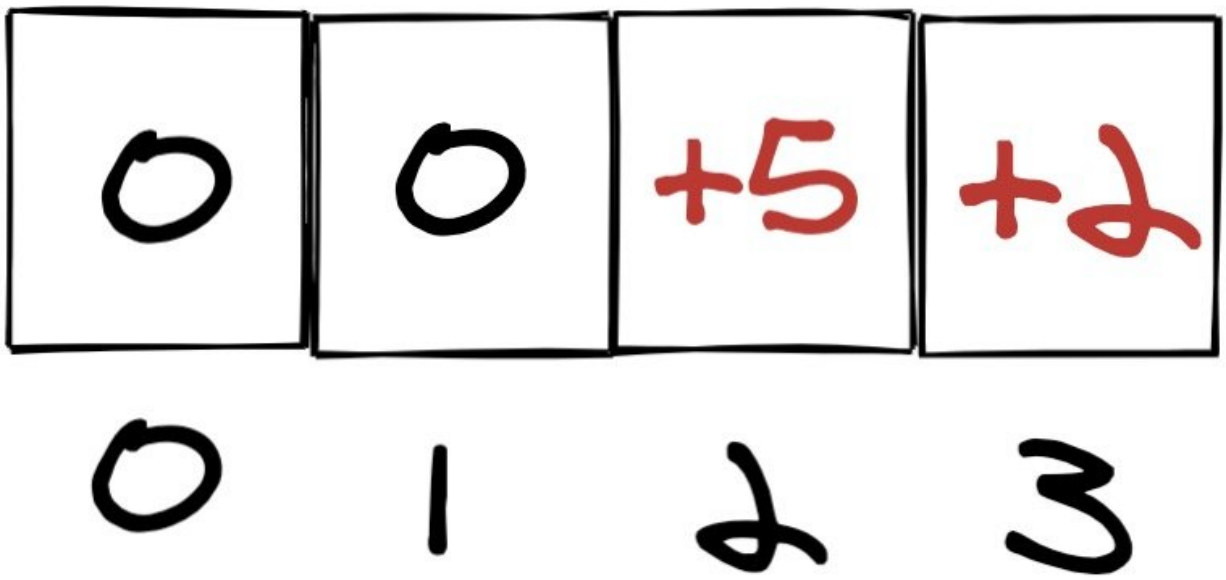




如图黄色部分是我们需要执行增加操作，我这里画了一个挡板分割，实际上这个挡板不存在。那么如何记录黄色部分的信息呢？我举个例子来说

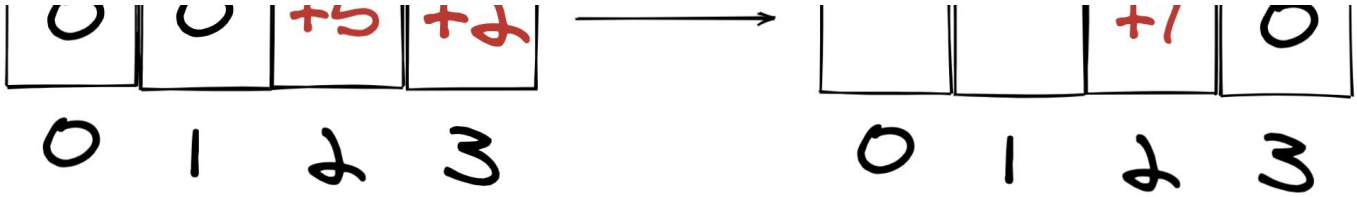
比如：

- 调用了 increment(3, 2)，就把 increment[3] 增加 2。
- 继续调用 increment(2, 5)，就把 increment[2] 增加 5。



而当我们 pop 的时候：

- 只需要将栈顶元素**加上 increment[cnt - 1]** 即可， 其中 cnt 为栈当前的大小。
- 另外，我们需要将 increment[cnt - 1] 更新到 increment[cnt - 2]，并将 increment[cnt - 1] 重置为 0。



代码

代码支持：Python，Java

Python Code：

```python
class CustomStack:

    def __init__(self, size: int):
        self.st = []
        self.cnt = 0
        self.size = size
        self.incrementals = [0] * size

    def push(self, x: int) -> None:
        if self.cnt < self.size:
            self.st.append(x)
            self.cnt += 1


    def pop(self) -> int:
        if self.cnt == 0: return -1
        if self.cnt >= 2:
            self.incrementals[self.cnt - 2] += self.incrementals[self.cnt - 1]
        ans = self.st.pop() + self.incrementals[self.cnt - 1]
        self.incrementals[self.cnt - 1] = 0
        self.cnt -= 1
        return ans


    def increment(self, k: int, val: int) -> None:
            if self.cnt:
                self.incrementals[min(self.cnt, k) - 1] += val
```

Java Code:

```java
class CustomStack {

    Stack<Integer> stack;
    int[] incrementals;
```

```
    int maxSize = -1;

    public CustomStack(int maxSize) {
        this.maxSize = maxSize;
        stack = new Stack<>();
        incrementals = new int[maxSize];
    }

    public void push(int x) {
        if (stack.size() < maxSize) {
            stack.push(x);
        }
    }

    public int pop() {
        int i = stack.size() - 1;
        if (i < 0) return -1;
        if (i > 0) {
            incrementals[i - 1] += incrementals[i];
        }
        int res = stack.pop() + incrementals[i];
        incrementals[i] = 0;
        return res;
    }

    public void increment(int k, int val) {
        int i = Math.min(k, stack.size()) - 1;
        if (i >= 0) {
            incrementals[i] += val;
        }
    }
}
```

复杂度分析

- 时间复杂度：全部都是 $O(1)$
- 空间复杂度：我们维护了一个大小为 maxSize 的数组，因此平均到每次的空间复杂度为 $O(\text{maxSize}/N)$，其中 N 为操作数。

优化的前缀和

思路

上面的思路无论如何，我们都需要维护一个大小为 $O(\text{maxSize})$ 的数组 incremental 。而由于栈只能在栈顶进行操作，因此这实际上可以稍微优化一点，即维护一个大小为当前栈长度的 incrementals，而不是 $O(\text{maxSize})$ 。

每次栈 push 的时候，incrementals 也 push 一个 0。每次栈 pop 的时候， incrementals 也 pop，这样就可以了。

> 这里的 incrementals 并不是一个栈，而是一个普通数组，因此可以随机访问。

代码

```
class CustomStack:

    def __init__(self, size: int):
        self.st = []
        self.cnt = 0
        self.size = size
        self.incrementals = []

    def push(self, x: int) -> None:
        if self.cnt < self.size:
            self.st.append(x)
            self.incrementals.append(0)
            self.cnt += 1


    def pop(self) -> int:
        if self.cnt == 0: return -1
        self.cnt -= 1
        if self.cnt >= 1:
            self.incrementals[-2] += self.incrementals[-1]
        return self.st.pop() + self.incrementals.pop()


    def increment(self, k: int, val: int) -> None:
        if self.incrementals:
            self.incrementals[min(self.cnt, k) - 1] += val
```

复杂度分析

- 时间复杂度：全部都是 $O(1)$
- 空间复杂度：我们维护了一个大小为 cnt 的数组，因此平均到每次的空间复杂度为 $O(cnt/N)$，其中 N 为操作数，cnt 为操作过程中的栈的最大长度（小于等于 maxSize）。

可以看出优化的解法在 maxSize 非常大的时候是很有意义的。

相关题目

- 155. 最小栈

更多题解可以访问我的 LeetCode 题解仓库：https://github.com/azl397985856/leetcode 。 目前已经 37K star 啦。

大家也可以关注我的公众号《力扣加加》获取更多更新鲜的 LeetCode 题解

欢迎长按关注

上一页 下一页