线性表

LeetCode刷题过程中，常常用到的线性表主要包括以下四个重要的数据结构: 数组、链表、栈、队列。

下面将分别讲解数组、链表、栈和队列。

线性表概述

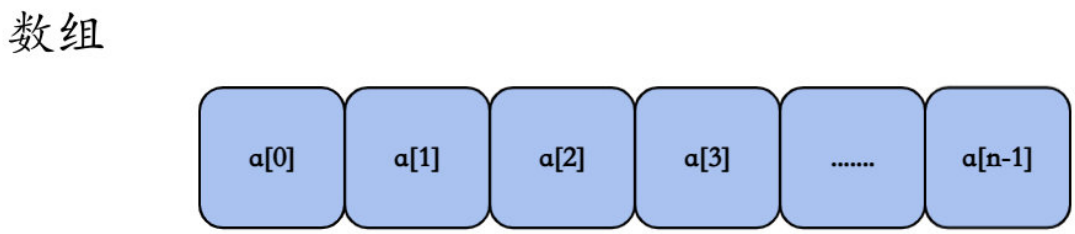
线性: 这里的线性是逻辑上的连续，而非物理存储的连续。

存储的数据: 线性表是一个有`n`个相同类型数据的有序序列。

数组array

介绍

数组是物理存储连续的线性表，其常见的形式为 `a[0]、a[1] ... a[n-1]`，`a[i-1]` 是 `a[i]` 的前驱，`a[i+1]` 是 `a[i]` 的后继。

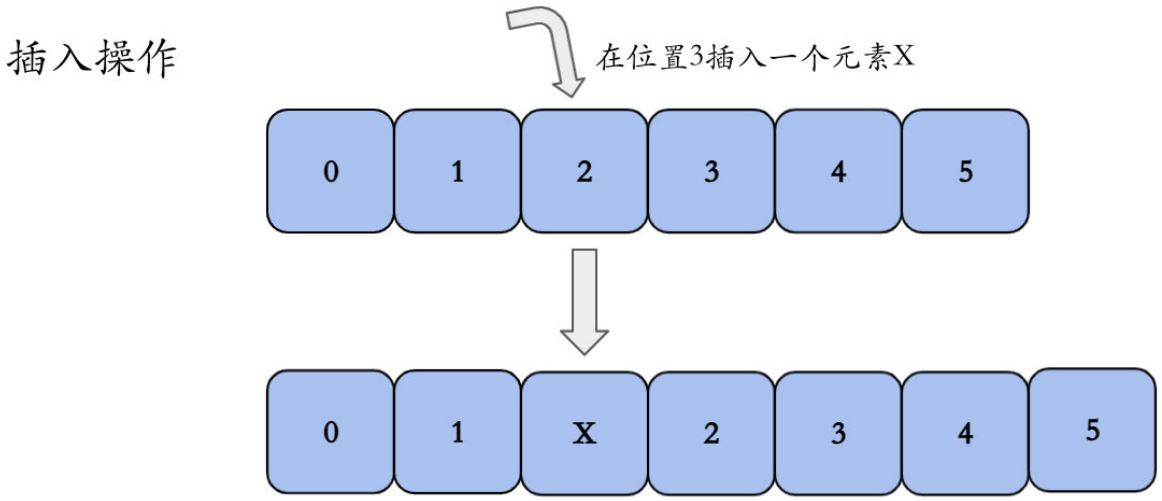


基本操作

插入

插入元素，要将插入位置后的元素全部向后移动一位。

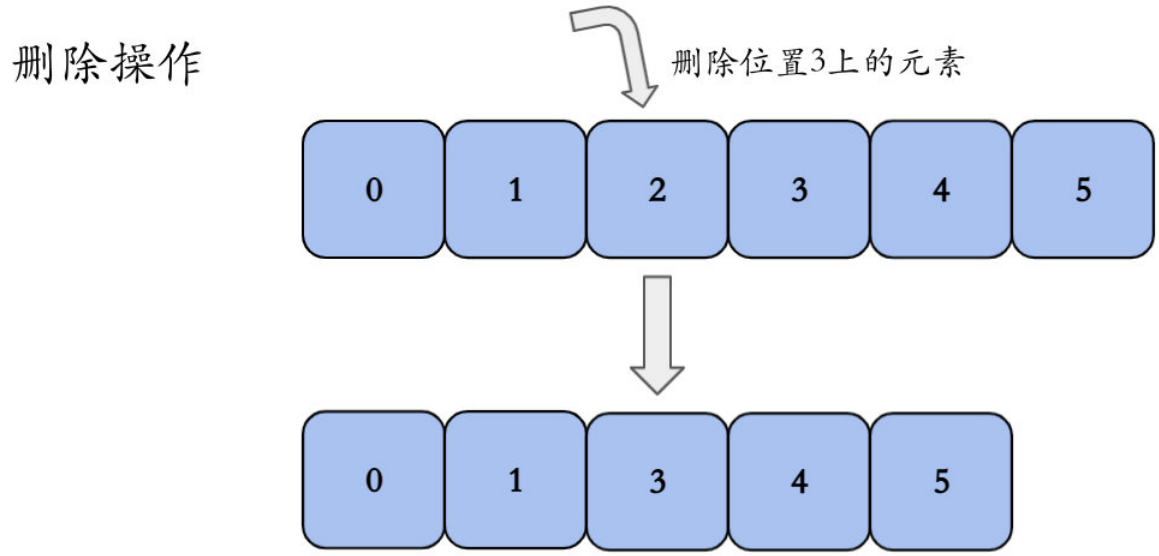
下图以数组长度为6，数据为`0、1、2、3、4、5`，在**位置3**插入一个元素X举例。



删除

删除元素，要讲删除位置后的元素全部向前移动一位。

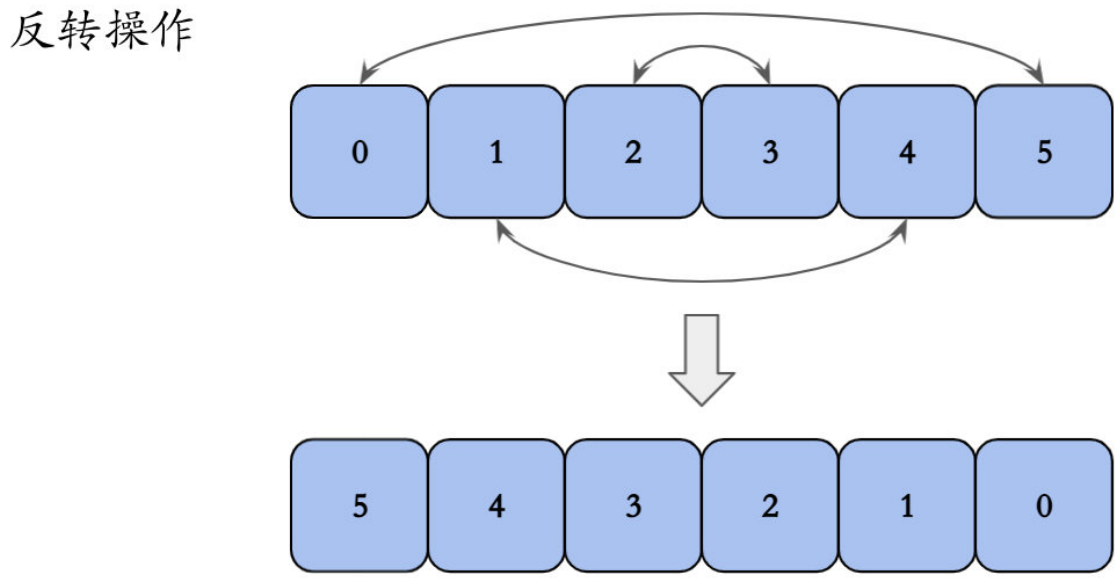
下图以数组长度为6，数据为`0、1、2、3、4、5`，删除**位置3**上的元素X举例。



反转

翻转数组，本质是将数组存储的数据进行反转。

下图以数组长度为6，数据为`0、1、2、3、4、5`，反转整个数组举例。



例题

LeetCode 27. 移除元素

题意

删除数组中所有等于 `val` 的元素，返回移除后数组的新长度。要求不使用额外的空间。

示例

```
输入：nums = [3,2,2,3], val = 3
输出：2, nums = [2,2]
```

题解

数组的删除操作，但如何不使用额外的空间呢？因为删除`val`后的数组的长度小于等于原数组的长度，因此可以一边将不等于`val`的数组放入原数组中，同时判断原数组的数是否等于`val`。

代码

```java
class Solution {
    public int removeElement(int[] nums, int val) {
        // left 存当前nums数组中不等于val的数字数量
        int left = 0;
        for (int right = 0; right < nums.length; right++) {
            if (nums[right] != val) {
                nums[left] = nums[right];
                left++;
            }
        }
        return left;
    }
}
```

习题推荐

LeetCode 35. 搜索插入位置

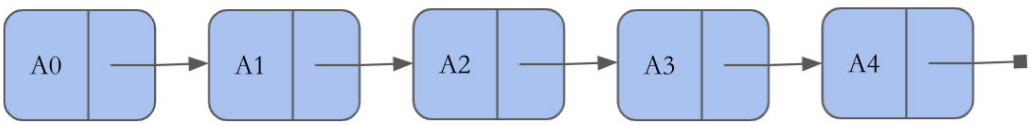
链表

介绍

链表的出现是为了解决数组插入、删除带来的线性开销。

区别于数组，链表中的元素可以不连续存储，每一个元素包含该 **元素的数据** 和 **指向链表下一个节点的指针**。

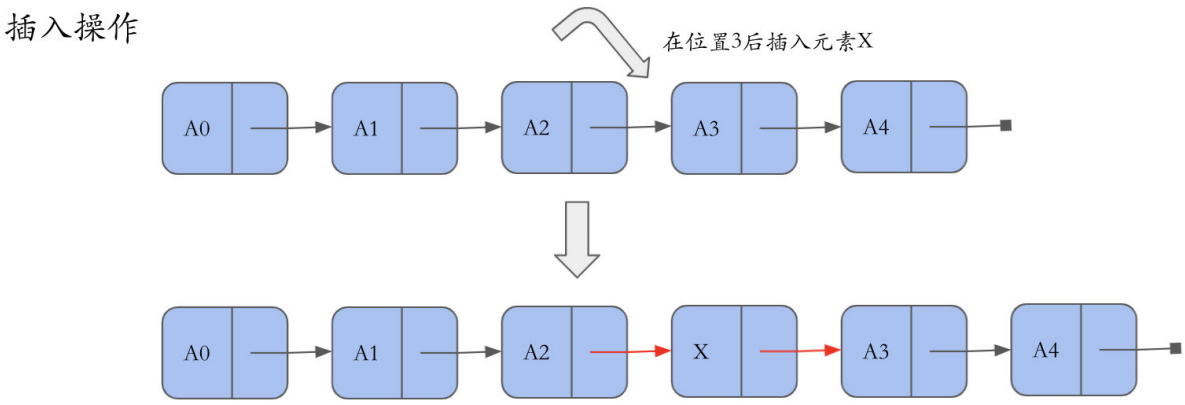
链表



基本操作

插入

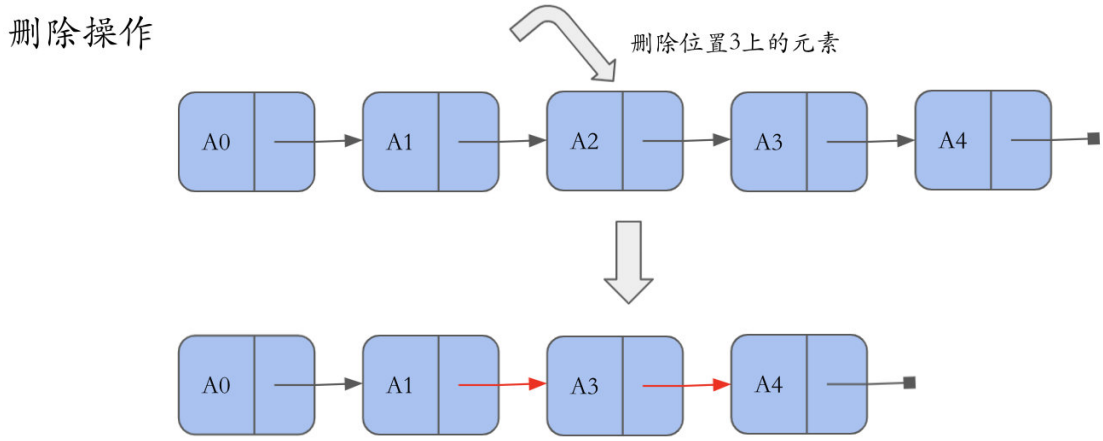
插入元素，要将插入元素前一个位置的指针指向插入元素本身，将插入元素的指针指向前一个位置。



删除

删除元素，要将 删除元素前一个元素的指针 指向 删除元素后一个元素，代码实现上需要将 删除元素指针指向的位置 记录下来。

下图是以长度为5的链表，删除位置3上的元素为例子。

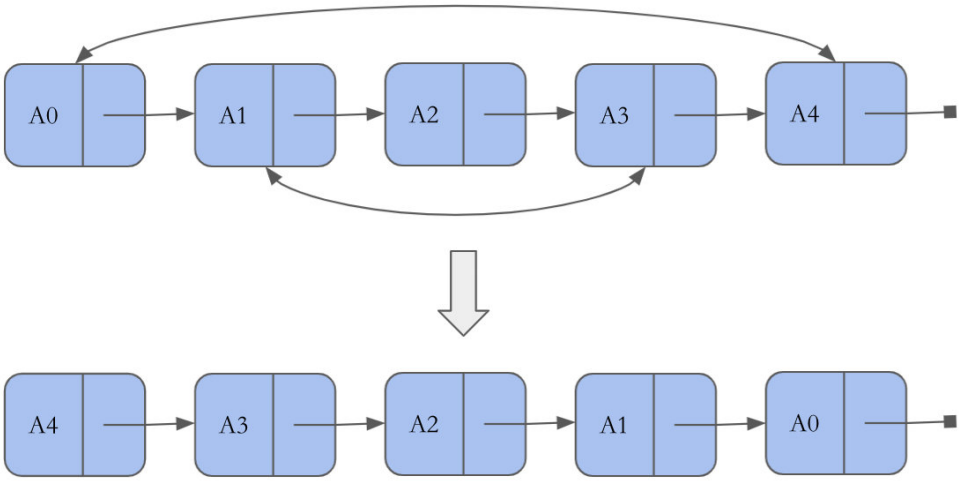


翻转

翻转链表，可以一边遍历一边用一个临时变量记录当前元素的下一个元素指针所指向的位置，然后再将当前元素的下一个元素指针指向自己。

下图是以长度为5的链表，翻转链表为例子。

翻转操作



例题

1. LeetCode 206. 反转链表

题意

给单链表的头节点 `head` ，请反转链表，并返回反转后的链表。

示例

```
输入：head = [1,2,3,4,5]
输出：[5,4,3,2,1]
```

题解

按上述链表翻转操作思路实现代码。

代码

```
public ListNode reverseList(ListNode head) {
    // pre 存的是当前节点的上一个节点
    ListNode prev = null;
    // curr 存的是当前链表遍历到节点
    ListNode curr = head;
    while (curr != null) {
        // next 存的是当前节点下一个节点
        ListNode next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
```

```
    return prev;
}
```

习题推荐

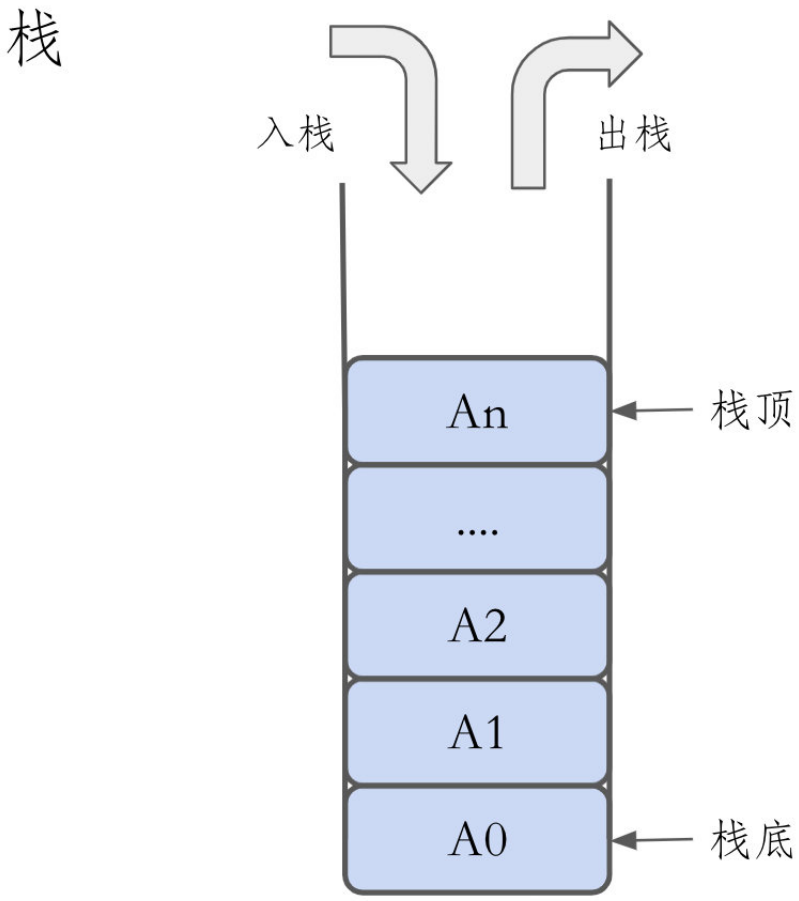
1. LeetCode237. 删除链表中的节点
2. LeetCode 21. 合并两个有序链表
3. LeetCode 160. 相交链表

栈和队列

栈介绍

栈被限定必须在栈顶进行插入和删除操作，因此其特点为是**后进先出**。

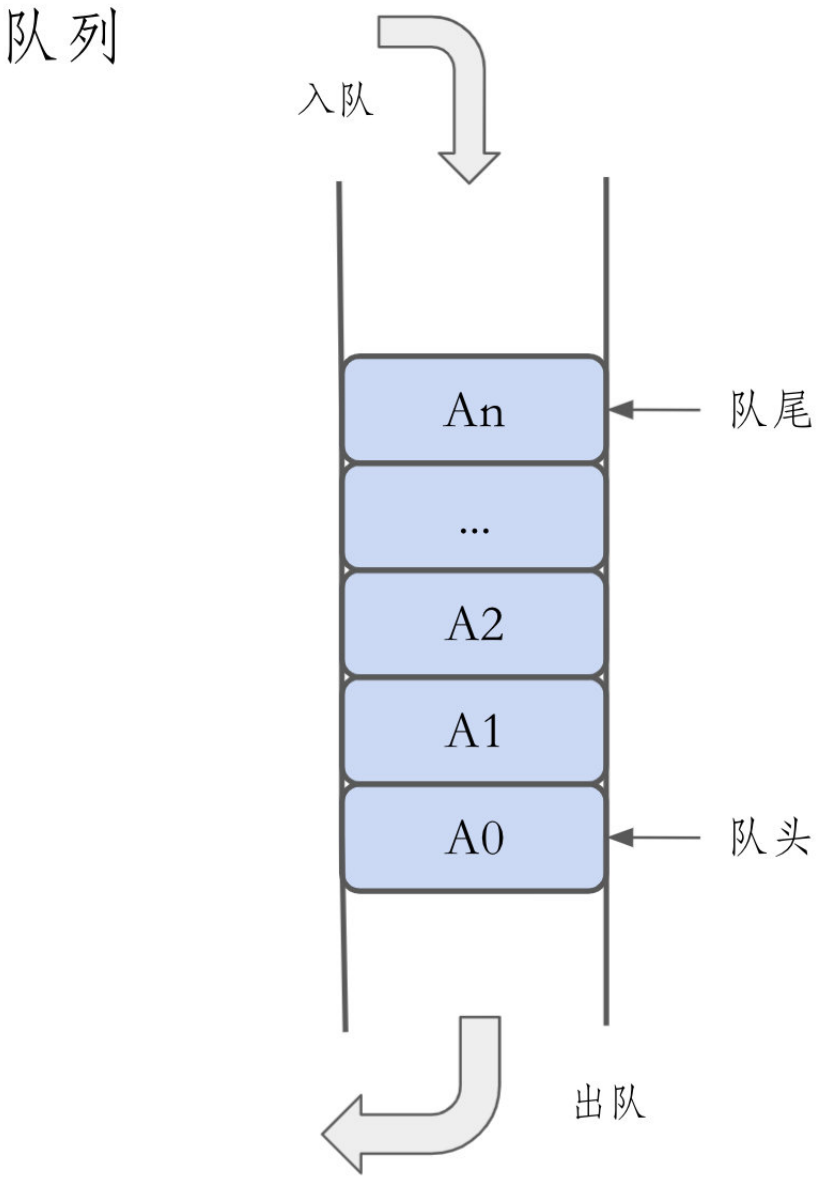
下图是栈的插入(入栈)、删除(出栈)示意图。



队列介绍

队列被限定在队头进行删除操作，队尾进行插入操作，因此其特点为**先进先出**。

下图是队列的插入(入队)、删除(出队)示意图。



基本操作

栈和队列的插入和删除操作上图已解释。

例题

LeetCode 155. 最小栈

题意 设计一个支持 `push` ， `pop` ， `top` 操作，并能在常数时间内检索到最小元素的栈。

- `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。

示例

```
输入：
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

输出：
[null,null,null,null,-3,null,0,-2]

解释：
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.getMin();   --> 返回 -2.
```

题解

新建辅助栈，辅助栈的栈顶表示原栈所有数字最小值，下面分别讨论题目要求的四种操作，分别如何实现。

- `push(x)`: 若插入的数字小于等于辅助栈的栈顶元素，则这个数字在原栈是最小值(之一)，我们将其插入辅助栈中。
- `pop()`: 若原栈删除的数字等于辅助栈的栈顶元素，则这个数字在原栈是最小值(之一)，我们同时原栈和辅助栈的栈顶元素；反之，只删除原栈栈顶元素。
- `top()`: 返回原栈的栈顶元素。
- `getMin()`: 返回辅助栈的栈顶元素。

代码

```
class MinStack {
```

```
    public Stack<Integer> s, min_s;
    public MinStack() {
        s = new Stack<>();
        min_s= new Stack<>();
    }
    public void push(int x) {
        s.push(x);
        if(min_s.isEmpty() || x <= min_s.peek())
            min_s.push(x);
    }
    public void pop() {
        if(s.pop().equals(min_s.peek()))
            min_s.pop();
    }
    public int top() {
        return s.peek();
    }
    public int getMin() {
        return min_s.peek();
    }
}
```

习题推荐

LeetCode 20. 有效的括号