# 题目

整数数组的一个 排列 就是将其所有成员以序列或线性顺序排列。

例如，arr = [1,2,3] ，以下这些都可以视作 arr 的排列：[1,2,3]、[1,3,2]、[3,1,2]、[2,3,1] 。

整数数组的 下一个排列 是指其整数的下一个字典序更大的排列。更正式地，如果数组的所有排列根据其字典顺序从小到大排列在一个容器中，那么数组的 下一个排列 就是在这个有序容器中排在它后面的那个排列。如果不存在下一个更大的排列，那么这个数组必须重排为字典序最小的排列（即，其元素按升序排列）。

例如，arr = [1,2,3] 的下一个排列是 [1,3,2] 。

类似地，arr = [2,3,1] 的下一个排列是 [3,1,2] 。

而 arr = [3,2,1] 的下一个排列是 [1,2,3] ，因为 [3,2,1] 不存在一个字典序更大的排列。

给你一个整数数组 nums ，找出 nums 的下一个排列。

必须原地修改，只允许使用额外常数空间。

示例 1：

输入：nums = [1,2,3]

输出：[1,3,2]

示例 2：

输入：nums = [3,2,1]

输出：[1,2,3]

示例 3：

输入：nums = [1,1,5]

输出：[1,5,1]

提示：

1 <= nums.length <= 100

0 <= nums[i] <= 100

# 分析

## 方法一：两遍扫描

思路：

代码：

class Solution {

public:

void nextPermutation(vector<int>& nums) {

int i = nums.size() - 2; // 遍历第一个数

// 从后向前找到第一个不满足递减的数

// 之所以从右侧开始遍历，是因为这里需要找下一个最大的数字

// 下一个肯定是从个位开始

while (i >= 0 && nums[i] >= nums[i + 1]) {

i--;

}

// 如果找到了这样的数

if (i >= 0) {

// 从后向前找到第一个比 nums[i] 大的数

int j = nums.size() - 1;

while (j >= 0 && nums[i] >= nums[j]) {

j--;

}

// 交换两个数

swap(nums[i], nums[j]);

}

// 将 i 后面的数逆序排列

reverse(nums.begin() + i + 1, nums.end());

}

};

复杂度分析

时间复杂度：O(N)，其中N为给定序列的长度。我们至多只需要扫描两次序列，以及进行一次反转操作。

空间复杂度：O(1)，只需要常数的空间存放若干变量。