## 概要

我们需要找到给定数字列表的下一个字典排列，而不是由给定数组形成的数字。

## 解决方案

#### 方法一：暴力法

****算法****

在这种方法中，我们找出由给定数组的元素形成的列表的每个可能的排列，并找出比给定的排列更大的排列。  
但是这个方法是一种非常天真的方法，因为它要求我们找出所有可能的排列  
这需要很长时间，实施起来也很复杂。  
因此，这种方法根本无法通过。 所以，我们直接采用正确的方法。

****复杂度分析****

* 时间复杂度：O(n!)O(n!)O(n!)，可能的排列总计有 n!n!n! 个。
* 空间复杂度：O(n)O(n)O(n)，因为数组将用于存储排列。

#### 方法二：一遍扫描

****算法****

首先，我们观察到对于任何给定序列的降序，没有可能的下一个更大的排列。

例如，以下数组不可能有下一个排列：

[9, 5, 4, 3, 1]

我们需要从右边找到第一对两个连续的数字 a[i]a[i]a[i] 和 a[i−1]a[i-1]a[i−1]，它们满足 a[i]>a[i−1]a[i]>a[i-1]a[i]>a[i−1]。现在，没有对 a[i−1]a[i-1]a[i−1] 右侧的重新排列可以创建更大的排列，因为该子数组由数字按降序组成。因此，我们需要重新排列 a[i−1]a[i-1]a[i−1] 右边的数字，包括它自己。

现在，什么样子的重新排列将产生下一个更大的数字呢？我们想要创建比当前更大的排列。因此，我们需要将数字 a[i−1]a[i-1]a[i−1] 替换为位于其右侧区域的数字中比它更大的数字，例如 a[j]a[j]a[j]。



我们交换数字 a[i−1]a[i-1]a[i−1] 和 a[j]a[j]a[j]。我们现在在索引 i−1i-1i−1 处有正确的数字。 但目前的排列仍然不是我们正在寻找的排列。我们需要通过仅使用 a[i−1]a[i-1]a[i−1]右边的数字来形成最小的排列。 因此，我们需要放置那些按升序排列的数字，以获得最小的排列。

但是，请记住，在从右侧扫描数字时，我们只是继续递减索引直到我们找到 a[i]a[i]a[i] 和 a[i−1]a[i-1]a[i−1] 这对数。其中，a[i]>a[i−1]a[i] > a[i-1]a[i]>a[i−1]。因此，a[i−1]a[i-1]a[i−1] 右边的所有数字都已按降序排序。此外，交换 a[i−1]a[i-1]a[i−1] 和 a[j]a[j]a[j] 并未改变该顺序。因此，我们只需要反转 a[i−1]a[i-1]a[i−1] 之后的数字，以获得下一个最小的字典排列。

下面的动画将有助于你理解：



* Java

public class Solution {

public void nextPermutation(int[] nums) {

int i = nums.length - 2;

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

i--;

}

if (i >= 0) {

int j = nums.length - 1;

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

j--;

}

swap(nums, i, j);

}

reverse(nums, i + 1);

}

private void reverse(int[] nums, int start) {

int i = start, j = nums.length - 1;

while (i < j) {

swap(nums, i, j);

i++;

j--;

}

}

private void swap(int[] nums, int i, int j) {

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

}

****复杂度分析****

时间复杂度：O(n)O(n)O(n)，在最坏的情况下，只需要对整个数组进行两次扫描。

空间复杂度：O(1)O(1)O(1)，没有使用额外的空间，原地替换足以做到。