26 下一个排列

更新时间：2019-09-10 09:53:06

> 读一本好书，就是和许多高尚的人谈话。
>
> ——歌德

刷题内容

> 难度: **Medium**
>
> https://leetcode-cn.com/problems/next-permutation/

题目描述

实现获取下一个排列的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列。

如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。

必须原地修改，只允许使用额外常数空间。

以下是一些例子，输入位于左侧列，其相应输出位于右侧列。
1,2,3 → 1,3,2
3,2,1 → 1,2,3
1,1,5 → 1,5,1

解题方案

思路：时间复杂度: O (N) 空间复杂度: O (1)

> *引子*：十进制数怎么求下一个数？

比如 1099，从最后一位 9 开始寻找，发现 9 不存在下一个数字，于是看倒数第二位，倒数第二位同样是 9 也不存在下一个数字。看倒数第 3 位，是 0，下一个数字是 1，于是把倒数第三位变成下一个数字 1，再把后面两位变成最小的两个数字，也就是 00，于是 1099 的下一个数就是 1100。

最后我们总结出来的步骤：

- 从后面开始，寻找第一个有下一个数字的数字，假设该位置为倒数第 i 位。我们把倒数第 i 位变成它的下一个数字
- 将 倒数第 1 位 - 第 i-1 位变成最小的数字，也就是 0

首先，什么是全排列呢？上过高中的同学肯定都知道，回顾一下：全排列

这里题目需要我们获得下一个更大的排列，那么我们想一下，在所有位元素固定的情况下，如何让这个排列数字更大呢？自然是把大一点的数字放在前面比较好啦。但由于我们只是去找下一个更大的排列，而不仅仅是找到一个更大的就行，所以我们要将最合适的一个较大的数字和前面一个较小的数字换一下位置。

- 如果一个序列是递减的，那么它不具有下一个排列。因为他就是最大的排列
- 如果一个序列是递增的，那么它是最小的排列

算法具体流程：

- 从后面开始，寻找第一个有下一个数字的数字，假设该位置为倒数第 i 位。这里要注意，由于从开头到倒数第 i+1 位，数字不变，因此倒数第 i 位能选的数字只能是倒数第 i 位后面的数字
- 将倒数第 i 位变成它的取值范围内的下一个数字
- 将倒数第 1 位 - 第 i-1 位变成最小的排列，也就是，数字升序（由于倒数第 i 位后面的数字是降序的，所以可以利用这个性质来实现 o (n) 的算法来实现升序）

我们来看一个具体的例子，一个排列为 124653

我们倒着往前找，遇到 4 小于 6 的情况停止，因为 4 后面有比它大的数。

并且我们可以知道：

1. 124653 和它的下一个排列的公共前缀为 12 (因为 4653 存在下一个排列，所以前面的数字 12 保持不变)
2. 4 后面的元素是递减的 (上面介绍的终止条件是前一个元素小于后一个元素，这里是 4<6)

现在，我们开始考虑如何找到 4653 的下个排列，首先明确 4 后面的几个数字中至少有一个大于 4.

4 肯定要和 653 这 3 个数字中大于 4 的数字中 (6，5) 的某一个进行交换。这里就是 4 要和 6，5 中的某一个交换，很明显要和 5 交换，如果找到这样的元素呢，因为我们知道 4 后面的元素是递减的，所以在 653 中从后面往前查找，找到第一个大于 4 的数字，这就是需要和 4 进行交换的数字。这里我们找到了 5，交换之后得到的临时序列为 5643.，交换后得到的 643 也是一个递减序列。

所以得到的 4653 的下一个临时序列为 5643，但是既然前面数字变大了 (4653—>5643)，后面的自然要变为升序才行，变换 5643 得到 5346.

所以 124653 的下一个序列为 125346.

再来一个例子，比如 125430

- 从末尾开始，找到 decreasing subsequence，5430，因为来调 5430 无论怎么调，都不可能有比它更小的，数也被自然的分成两部分 (1,2) 和 （5，4，3，0)
- 下一步是找这个 sequence 里面第一个比前面部分，比 2 大的，3，也很容易理解，因为下一个必定是 (1,3) 打头
- 交换 3 和 2 ，变成 (1,3,5,4,2,0), 再把后面的部分 reverse，得到后面部分可得到的最小的

这个时候，得到下一个 sequence 130245

下面我们来看下具体的代码实现：

python beats 99.89%

```
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        if len(nums) <= 1:
            return
        idx = 0
        for i in range(len(nums)-1, 0, -1): # 从后往前看
            if nums[i] > nums[i-1]: # 找到第一个比后面元素小的元素nums[i]
                idx = i
                break
        if idx != 0: # 如果这个元素不是第一个元素，说明整个num不是递减序列
            for i in range(len(nums)-1, idx-1, -1): # 把nums[i]和它后面最后一个大于它的元素交换一下位置
                if nums[i] > nums[idx-1]:
                    nums[i], nums[idx-1] = nums[idx-1], nums[i]
                    break

        nums[idx:] = nums[idx:][::-1] # 后面的元素都需要变成升序才行
```

c++ beats 98.83%

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int idx = (int)nums.size() - 2;
        //寻找从右边开始第一个比右边小的数
        while (idx >= 0 && nums[idx] >= nums[idx + 1]) {
            idx--;
        }
        if (idx >= 0) {
            int next = (int)nums.size() - 1;
            //寻找nums[idx]的下一个数
            while (next >= 0 && nums[next] <= nums[idx]) {
                next--;
            }
            //交换nums[idx]和nums[next]
            int temp = nums[idx];
            nums[idx] = nums[next];
            nums[next] = temp;
        }
        //将nums[idx+1..]倒过来
        int left = idx + 1, right = (int)nums.size() - 1;
        while (left < right) {
            int temp = nums[left];
            nums[left] = nums[right];
            nums[right] = temp;
            left++;
            right--;
        }
    }
};
```

java beats 89.89%

```
class Solution {
    public void nextPermutation(int[] nums) {
        int idx = nums.length - 2;
        //寻找从右边开始第一个比右边小的数
        while (idx >= 0 && nums[idx] >= nums[idx + 1]) {
            idx--;
        }
        if (idx >= 0) {
            int next = nums.length - 1;
            //寻找nums[idx]的下一个数
            while (next >= 0 && nums[next] <= nums[idx]) {
                next--;
            }
            //交换nums[idx]和nums[next]
            int temp = nums[idx];
            nums[idx] = nums[next];
            nums[next] = temp;
        }
        //将nums[idx+1..]倒过来
        int left = idx + 1, right = nums.length - 1;
        while (left < right) {
            int temp = nums[left];
            nums[left] = nums[right];
            nums[right] = temp;
            left++;
            right--;
        }
    }
}
```

go beats 100%

```
func nextPermutation(nums []int) {
 n := len(nums)
 idx := n - 2;
 //寻找从右边开始第一个比右边小的数
 for idx >= 0 && nums[idx] >= nums[idx + 1] {
  idx--;
 }
 if idx >= 0 {
  next := n - 1;
  //寻找nums[idx]的下一个数
  for next >= 0 && nums[next] <= nums[idx] {
   next--;
  }
  //交换nums[idx]和nums[next]
  temp := nums[idx];
  nums[idx] = nums[next];
  nums[next] = temp;
 }
 //将nums[idx+1..]倒过来
 left := idx + 1
 right := n - 1;
 for left < right {
  temp := nums[left];
  nums[left] = nums[right];
  nums[right] = temp;
  left++;
  right--;
 }
}
```

小结

这道题的难点在于如何十进制数怎么求下一个数？同学们吧过程仔细的多看几遍，然后动手敲，敲第一遍的时候不要看代码，硬着头皮上。这样会对你的算法能力有很大的促进。

}

25 盛水最多的容器

27 搜索旋转排序数组