给定一个不存在重复元素的数组，要求得到所有该数组的排列方案。

思路1：也是一个全排列模板。由于全排列共有n!种方案，则对于每个数字，出现在任一位置的方案数为(n-1)!，并且n个元素的排列方案，可以看做n-1个元素的排列方案上再加上一个可以放在任意位置的第n个元素。即此问题具有子问题，但第n个元素放在什么位置是需要根据当前方案生成的。最简单的做法是，将每个元素都放在第一位上，然后生成剩余的n-1位，在n-1位生成完成后，继续讲下一个元素放在第一位。准确性：首先将第1个元素放在第1位，生成其余n-1位数字的全排列，然后互换第1和第2个数字，此时之前的第2位位于第1位，继续生成其余n-1位数字的全排列（此时相比之前的情况，将第1位数字放入，第二位数字取出，从而得到不同的解），如此类推直到第n位放到第1位。而在生成n-1位数字的全排列时又变成生成n位数字的子问题，依旧可以使用这一的方法，直到剩余0位数字的排列需要生成。

def dfs(pos):

if pos == len(nums):

ans.append(nums[:])

for i in range(pos, len(nums)):

nums[i], nums[pos] = nums[pos], nums[i]

dfs(pos + 1)

nums[i], nums[pos] = nums[pos], nums[i]

ans = []

dfs(0)

return ans

思路2：不使用替换数字的方式，即对输入不做任何修改，而是使用一个标记数组来记录哪一位置的数字被使用过。在生成排列的下一个数字时，取出第一个没有被标记使用过的数字，然后标记其为已使用，并继续递归，直到所有数字均使用过生成可行解。

def dfs(sol):

            if len(sol) == len(nums):

                ans.append(sol[:])

                return

            for i in range(len(nums)):

                if flag[i] == 0:

                    flag[i] = 1

                    dfs(sol + [nums[i]])

                    flag[i] = 0

        flag = [0] \* len(nums)

        ans = []

        dfs([])

        return ans

上述问题是不包含重复元素的全排列，此外还有具有重复元素的全排列。

思路1：交换的思想依然可以实现这个问题。交换要做的是将后续所有未使用的数字均放到当前位置一次，这里的未使用是指索引。而在包含重复元素的问题中，未使用表示的是数字本身，所以只需要记录当前位置放置过的数字，此后如果出现该数字就不再需要考虑即可。

def dfs(pos):

            if pos == len(nums):

                ans.append(nums[:])

            flag = set()

            for i in range(pos, len(nums)):

                if nums[i] not in flag:

                    flag.add(nums[i])

                    nums[i], nums[pos] = nums[pos], nums[i]

                    dfs(pos + 1)

                    nums[i], nums[pos] = nums[pos], nums[i]

        ans = []

        dfs(0)

        return ans

思路2：首先，对具有重复元素的数组做剪枝，通常都是先排序，这样将所有相同的元素聚集到一起，方便比较（只需要和前一位比较）。

剪枝条件为，当当前元素已经使用过剪枝，以及当前元素未使用，但和上一个元素值相同，且上一个元素未使用，剪枝。对于第二个剪枝情况这里做一下分析，首先和前一个元素相同，就需要做去重，如果前一个元素未使用，则说明当前位置已经填充过前一个元素，现在是回溯阶段，因为如果第一次进入递归前一个元素肯定是使用过的，而当前元素已经填充过前一个元素，此时再放入一个相同的元素会产生重复解，需要剪枝。

举例说明：[1,2,2’]，第一次得到的排列结果为[1,2,2’]，回溯到flag=[1,0,0]的状态时，发现flag[2]=0，并且2’==2且2当前并未使用，表明该位置是已经填充过2后回溯到这个状态的，此时如果再填2’，则此前已经得到过[1,2]这个解，产生重复。

def dfs(sol):

if len(sol) == len(nums):

ans.append(sol[:])

return

for i in range(len(nums)):

if flag[i] == 1:

continue

if i > 0 and nums[i] == nums[i - 1] and flag[i - 1] == 0:

continue

flag[i] = 1

dfs(sol + [nums[i]])

flag[i] = 0

nums.sort()

flag = [0] \* len(nums)

ans = []

dfs([])

return ans