LC 47 permutation with duplicate elements <Medium>

这题对我来说有些难，我没有太想明白.

我开始的思路： 用 permutation 排列组合的规则：

将 k至n的元素排序，有 (n-k+1)! 种排法

再将 k-1 个元素插入每种排法中，有 n-k+1 种插法.

这个思路用在 LC 46：permutation without duplicate elements，没问题. 利用 list 的 insert 方法.

只是不知可不可以用 insert(-1, element) 代替 append()

但是尝试用到这个题目中却遇到了困难：

思路：先将数组排序，如此便于跳过 duplicate number.

然后，如果 nums[k-1] == nums[k]，则在 n+1-k 个subset中，只插到将重复元素右侧.

代码见下页：

```
def permuteUnique(self, nums):
    if not nums or len(nums)==0: return [[]]
    res = []
    return self.helper(nums, res, 0)

def helper(self, nums, res, start):
    if start == len(nums)-1:
        return [[nums[start]]]
    temp_list = self.helper(nums, res, start+1)
    for ls in temp_list:
        for j in range(len(ls)):
            if nums[start] == ls[j]:
                continue
            temp = list(ls)
            temp.insert(j, nums[start])
            res.append(temp)
    # 此外，要在末尾亦可插
    ls.append(nums[start])
    res.append(ls)

return res
```

致命问题是有重复值出现，例过来排列也是一样的。
如 [1,1,2]： temp_list = [[1,2],[2,1]]，再插一个1...

目前大家用的方法依然是 backtrack 方法.

思路:

condition 1: len(temp) == len(nums):

可以 append 进 result 了.

condition 2: 用一个 used 数组记录使用情况.

所有两种记录:

① 当前值与它在 nums 中前一个值相等, 且它前一个值还未使用. 那不可用这个元素

代码:

```
if i>0 and nums[i] == nums[i-1] and not used[i-1]:
    continue
```

② 与①相反: 当前值与它在 nums 中前一个值相等, 且它前一个值已经使用过, 那不可用这个元素.

代码:

```
if i>0 and nums[i] == nums[i-1] and used[i-1]:
    continue
```

显然前者, ①, 更 intuitive. 这里不是 insert 在中间. 而只是单纯 append.

什么时候轮到 i 3，但是 i-1 未使用？

必是在子层里用了 i-1 后，完成 res.append(temp) 后，再将 used[i-1] 设为 False.

也就是说，在当前长度 append [nums[i]]，将会产生 append [nums[i-1]] 相同的结果.

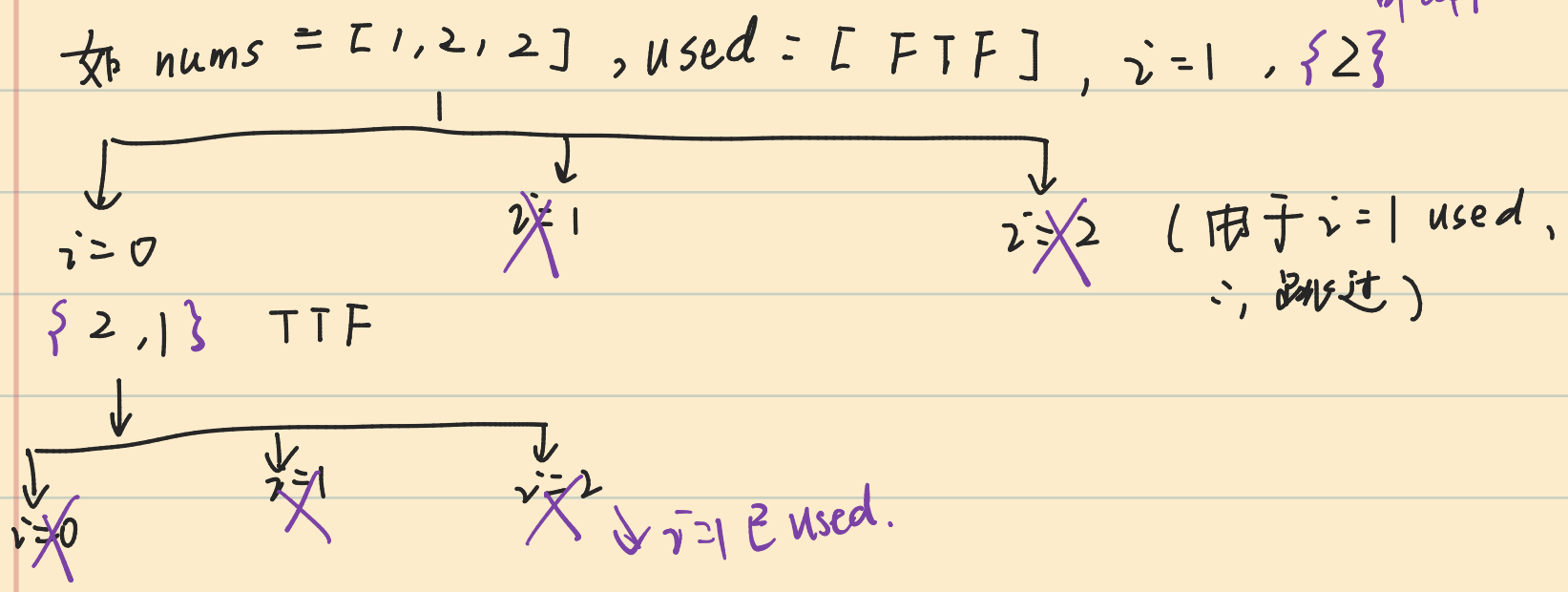
所以 hold on，此位置应放别的元素.

此后，若 nums[i+1] == nums[i] (== nums[i-1])

由于 used[i] == False，nums[i+1] 也会被跳过.

据称，方法2在某些位置无法 reach basecase，所以 less optimal.

递归的 ↓ 即 length 达到长度，可 append 了.



condition 3：每个元素用时，used[i] 标记为 True.

用完了，(从 backtrack 中出栈了)，再标为 False，以便下次使用.

code:

```
def permuteUnique(self, nums):
    if not nums or len(nums) == 0:
        return [[]]
    res = []
    nums.sort()
    used = [False] * len(nums)
    self.backtrack(nums, res, [], used)
    return res

def backtrack(self, nums, res, temp, used):
    if len(temp) == len(nums):
        res.append(list(temp))
    else:
        for i in range(len(nums)):
            if used[i] or (i>0 and nums[i-1] == nums[i]
                           and not used[i-1]): continue
            used[i] = True
            temp.append(nums[i])
            self.backtrack(nums, res, temp, used)
            used[i] = False
            temp.pop()
```

不sort怎么skip掉duplicates!