回溯法：模板、套路、简化写法

[负雪明烛](https://leetcode-cn.com/u/fuxuemingzhu/)

发布于 2021-03-3016.1k[递归](https://leetcode-cn.com/tag/recursion/)[C++](https://leetcode-cn.com/topic/cpp/)[Python](https://leetcode-cn.com/topic/python/)[Python3](https://leetcode-cn.com/topic/python3/)

各位题友大家好！ 今天是 [**@负雪明烛**](https://leetcode-cn.com/u/fuxuemingzhu/) 坚持日更的第 **66** 天。今天力扣上的每日一题是「[90. 子集 II](https://leetcode-cn.com/problems/subsets-ii/" \t "_blank)」。

**解题思路**

这个题是 [78. 子集](https://leetcode-cn.com/problems/subsets/) 的变形。

**回溯法**

一般情况下，看到题目要求「所有可能的结果」，而不是「结果的个数」，我们就知道需要暴力搜索所有的可行解了，可以用「回溯法」。

「回溯法」实际上一个类似枚举的搜索尝试过程，主要是在搜索尝试过程中寻找问题的解，当发现已不满足求解条件时，就「回溯」返回，尝试别的路径。

**回溯法**是一种算法思想，而**递归**是一种编程方法，回溯法可以用递归来实现。

回溯法的整体思路是：搜索每一条路，每次回溯是对具体的一条路径而言的。对当前搜索路径下的的未探索区域进行搜索，则可能有两种情况：

1. 当前未搜索区域满足结束条件，则保存当前路径并退出当前搜索；
2. 当前未搜索区域需要继续搜索，则遍历当前所有可能的选择：如果该选择符合要求，则把当前选择加入当前的搜索路径中，并继续搜索新的未探索区域。

上面说的未搜索区域是指搜索某条路径时的未搜索区域，并不是全局的未搜索区域。

回溯法搜所有可行解的模板一般是这样的：

res = []

path = []

def backtrack(未探索区域, res, path):

if path 满足条件:

res.add(path) # 深度拷贝

# return # 如果不用继续搜索需要 return

for 选择 in 未探索区域当前可能的选择:

if 当前选择符合要求:

path.add(当前选择)

backtrack(新的未探索区域, res, path)

path.pop()

backtrack 的含义是：未探索区域中到达结束条件的所有可能路径，path 变量是保存的是一条路径，res 变量保存的是所有搜索到的路径。所以当「未探索区域满足结束条件」时，需要把 path 放到结果 res 中。  
path.pop() 是啥意思呢？它是编程实现上的一个要求，即我们从始至终只用了一个变量 path，所以当对 path 增加一个选择并 backtrack 之后，需要清除当前的选择，防止影响其他路径的搜索。

**提醒**

下面的**正规写法**对应了全局共享一个 path 的写法，**简化写法**是每次新建 path。**这里是一个容易出错的细节**。

* 正规写法中全局共享一个 path，每次添加到 res 中，需要深度拷贝。python中是需要 copy.deepcopy(path) ；C++ 的 vector 的 push\_back() 函数，本身就是深度拷贝。另外由于是全局的 path，因此每次需要对 path 进行 push 和 pop 操作，
* 简化写法的 Python 解法在每次搜索位置区域的时候，使用的是产生一个新数组 path + [s[:i]] ，这样好处是方便：不同的路径使用的是不同的 path，因此不需要 path.pop() 操作；而且 res.append(path) 的时候不用深度拷贝一遍 path。

**一：求不含重复元素的数组的子集**

**正规写法**

对于 [78. 子集](https://leetcode-cn.com/problems/subsets/) 而言，找出没有重复数字的数组所有子集，按照模板，我们的思路应该是这样的：

1. **未探索区域**：剩余的未搜索的数组 nums[index: N - 1] ；
2. **每个 path 是否满足题目的条件**： 任何一个 path 都是子集，都满足条件，都要放到 res 中 ;
3. **当前 path 满足条件时，是否继续搜索**：是的，找到 nums[0:index-1] 中的子集之后， nums[index] 添加到老的 path 中会形成新的子集。
4. **未探索区域当前可能的选择**：每次选择可以选取 s 的 1 个字符，即 nums[index] ；
5. **当前选择符合要求**：任何 nums[index] 都是符合要求的，直接放到 path 中；
6. **新的未探索区域**：nums 在 index 之后的剩余字符串， nums[index + 1 : N - 1] 。

上面分析了那么多，让我们看看代码怎么写。由于每个path都是一个子集，都满足条件，所以就省略了一个 if 判断，并且由于当前 path 满足条件时，应该继续搜索，所以也省略了 if 中的 return 语句。

* Python
* C++

class Solution(object):

def subsets(self, nums):

res, path = [], []

self.dfs(nums, 0, res, path)

return res

def dfs(self, nums, index, res, path):

res.append(copy.deepcopy(path))

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

path.append(nums[i])

self.dfs(nums, i + 1, res, path)

path.pop()

我们发现代码比讲解简单。 for 循环的意义是说，从后续元素 nums[index:N-1] 中挑选剩余元素的时候，每个元素都有选和不选两个状态。

**简化写法**

上面是正规的回溯写法，如果想偷懒，可以每次在搜索的时候都新建一个 path 变量，而不是复用全局的 path。那么代码可以更精简。

如下面所写，每次寻找新的子集的时候，都新建了一个 path，因为 path + [nums[i]] 返回的是一个新的列表，放在函数的参数里面，每次传过来的都是新的，所以 res.append(path) 时不用深度拷贝。

* Python

class Solution(object):

def subsets(self, nums):

res = []

self.dfs(nums, 0, res, [])

return res

def dfs(self, nums, index, res, path):

res.append(path)

for i in xrange(index, len(nums)):

self.dfs(nums, i + 1, res, path + [nums[i]])

**二：求包含重复元素的数组的子集**

**正规写法**

如果你能理解上面的回溯法，那么包含重复元素的数组的子集，只不过一个小的改进。

比如说求 nums = [1,2,2] 的子集，那么对于子集 [1,2] 是选择了第一个 2，那么就不能再选第二个 2 来构成 [1,2] 了。所以，此时的改动点，就是先排序，每个元素 nums[i] 添加到 path 之前，判断一下 nums[i] 是否等于 nums[i - 1] ，如果相等就不添加到 path 中。

* Python
* C++

class Solution(object):

def subsetsWithDup(self, nums):

res, path = [], []

nums.sort()

self.dfs(nums, 0, res, path)

return res

def dfs(self, nums, index, res, path):

res.append(copy.deepcopy(path))

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

if i > index and nums[i] == nums[i - 1]:

continue

path.append(nums[i])

self.dfs(nums, i + 1, res, path)

path.pop()

**简化写法**

* python

class Solution(object):

def subsetsWithDup(self, nums):

res = []

nums.sort()

self.dfs(nums, 0, res, [])

return res

def dfs(self, nums, index, res, path):

if path not in res:

res.append(path)

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

if i > index and nums[i] == nums[i - 1]:

continue

self.dfs(nums, i + 1, res, path + [nums[i]])

**刷题心得**

我很喜欢刷回溯法的题目，因为模板是类似的。

参考资料：  
[90. Subsets II 解题报告](https://leetcode-cn.com/link/?target=https%3A%2F%2Fblog.csdn.net%2Ffuxuemingzhu%2Farticle%2Fdetails%2F79785548)  
[78. Subsets 解题报告](https://leetcode-cn.com/link/?target=https%3A%2F%2Fblog.csdn.net%2Ffuxuemingzhu%2Farticle%2Fdetails%2F79359540)  
[【LeetCode】代码模板，刷题必会](https://leetcode-cn.com/link/?target=https%3A%2F%2Ffuxuemingzhu.blog.csdn.net%2Farticle%2Fdetails%2F101900729)