

回溯法：

```
1 # 回溯法
2     result = []
3     def backtrack(路径, 选择列表):
4         if 满足结束条件:
5             result.add(路径)
6             return
7         for 选择 in 选择列表:
8             做选择
9             backtrack(路径, 选择列表)
10            撤销选择
```

1 全排列：

给定一个 没有重复 数字的序列，返回其所有可能的全排列。

示例：

```
输入：[1,2,3]
输出：
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

```
1 # def backtrack(路径, 选择列表): 路径path, 选择列表 nums
2 def dfs(nums, size, depth, used, path, res):
3     # nums: 要进行全排列的序列
4     # size: 序列大小
5     # depth : 第几层
6     # used: 序列, 记录当前数是否被使用
7     # path: 当前的选择除了的路径, 就是一种选择结果
8     # res: 结果
9     # 回溯终结条件: 把路径添加到结果
10    # 层数 == 序列大小, 说明已经到了树的最底层, 叶子节点, 把路径加入到结果
11    if depth == size:
12        res.append(path[:])
13        return
14    # for 选择 in 选择列表
15    # 从列表中做出选择
16    for i in range(size):
17        # 做选择
```

```

18         # 如果第i个数，没有添加过，就把这个数添加进了
19         if not used[i]:
20             used[i] = True
21             path.append(nums[i])
22
23             # 在下一层去做选择，深度+1
24             dfs(nums, size, depth+1, used, path, res)
25
26         # 撤回选择
27         used[i] = False
28         path.pop()

```

47. 全排列 II

难度 中等 371 371 371 371 371 371

给定一个可包含重复数字的序列，返回所有不重复的全排列。

示例:

```

输入: [1,1,2]
输出:
[
  [1,1,2],
  [1,2,1],
  [2,1,1]
]

```

```

1 # 分析，全排列II和全排列的区别在于，全排列中有重复元素，那么先按全排列写出代码，除去重
2 def permuteUnique(self, nums):
3     """
4     :type nums: List[int]
5     :rtype: List[List[int]]
6     """
7     def dfs(nums, size, depth, path, used, res):
8         if depth == size:
9             res.append(path[:])
10            return
11        for i in range(size):
12            if not used[i]:
13                # 在图中 ① 处，搜索的数也和上一次一样，但是上一次的 1 刚刚被
14                if i > 0 and nums[i] == nums[i - 1] and not used[i - 1]:
15                    continue
16
17                used[i] = True
18                path.append(nums[i])
19                dfs(nums, size, depth + 1, path, used, res)

```

```

20         used[i] = False
21         path.pop()
22
23     size = len(nums)
24     if size == 0:
25         return []
26
27     # 注意：这里是关键，需要对nums进行一次排列，这样，如果后面的数再次出现，就可
28     nums.sort()
29
30     used = [False] * len(nums)
31     res = []
32     dfs(nums, size, 0, [], used, res)
33     return res

```

39 组合总和

给定一个无重复元素的数组 `candidates` 和一个目标数 `target`，找出 `candidates` 中所有可以使数字和为 `target` 的组合。

`candidates` 中的数字可以无限重复被选取。

说明：

- 所有数字（包括 `target`）都是正整数。
- 解集不能包含重复的组合。

示例 1：

```

输入：candidates = [2,3,6,7], target = 7,
所求解集为：
[
  [7],
  [2,2,3]
]

```

示例 2：

```

输入：candidates = [2,3,5], target = 8,
所求解集为：
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]

```

解题思路如下图：

因为 `target = 7`，如果第一个先减2，找到 $7 - 2 = 5$ 的所有组合，那么 +2，就是7的组合

如果第一个先减3，找到 $7 - 3 = 4$ 的所有组合，在+3，那就是7的组合

依次类推

说明这个可以用回溯法

1 确定递归终止条件：

因为 `target` 一直减序列中的数，得到的差值就是新的 `target`，那么一直到 `target == 0`（图中红色），在结果中添加那条路径

```

1 if target == 0:

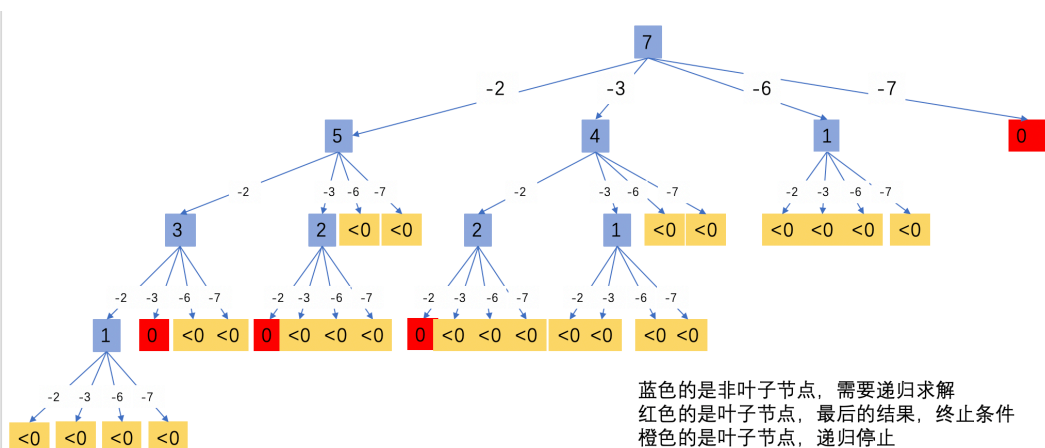
```

2 做递归求解，图中蓝色部分，就是target不为 0 的时候，从列表中找到要减的数

```

1  for i in range(begin, size):
2      # [2, 3, 6, 7]
3          residue = target - candidates[i]
4          # 第二个终止的可能，那就是，差值比0小，也不用循环了
5          if residue < 0 :
6              break
7          # 把对应减的数添加到path里
8          path.append(candidates[i])
9          # 回溯 这里的begin 换成了 i, [3, 6, 7],
10         backtrack(candidates, path, i, size, residue, res )
11
12     path.pop()

```



```

1 def combinationSum(self, candidates, target):
2     """
3     :type candidates: List[int]
4     :type target: int
5     :rtype: List[List[int]]
6     """
7     size = len(candidates)
8     if size < 0 :
9         return
10    res = []
11    path = []
12    candidates.sort()

```

```

13         def dfs(path, begin, target):
14             # 终止条件: target == 0
15             if target == 0:
16                 res.append(path[:])
17                 return
18             for i in range(begin, size):
19                 residue = target - candidates[i]
20                 if residue < 0:
21                     break
22                 path.append(candidates[i])
23                 dfs(path, i, residue)
24                 path.pop()
25
26         dfs(path, 0, target)
27         return res

```

40. 组合总和 II

难度 中等  331     

给定一个数组 `candidates` 和一个目标数 `target`，找出 `candidates` 中所有可以使数字和为 `target` 的组合。

`candidates` 中的每个数字在每个组合中只能使用一次。

说明：

- 所有数字（包括目标数）都是正整数。
- 解集不能包含重复的组合。

示例 1:

```

输入: candidates = [10,1,2,7,6,1,5], target = 8,
所求解集为:
[
  [1, 7],
  [1, 2, 5],
  [2, 6],
  [1, 1, 6]
]

```

示例 2:

```

输入: candidates = [2,5,2,1,2], target = 5,
所求解集为:
[
  [1,2,2],
  [5]
]

```

```

1 #组合2 和组合的区别在于，组合2的每个数字在每个组合只能用一次，所以，我就想直接在在原来
2 即把 dfs(path, i, residue) == >> dfs(path, i+1, residue) --> 这样就能保证每个
3 但是，不完全对，因为说明中还要求，解集中不包含重复组合
4 candidates全排列后，当candidates[i] == candidates[i-1]: 退出此次循环，说明这个
5
6 def combinationSum2(self, candidates, target):
7     """
8     :type candidates: List[int]
9     :type target: int
10    :rtype: List[List[int]]
11    """
12    size = len(candidates)
13    if size < 0 :
14        return
15    res = []
16    path = []
17    candidates.sort()
18    def dfs(path, begin, target ):
19        # 终止条件: target == 0
20        if target == 0:
21            res.append(path[:])
22            return
23
24        for i in range(begin, size):
25            residue = target - candidates[i]
26            if residue < 0 :
27                break
28            if i > begin and candidates[i] == candidates[i-1]:
29                continue
30            path.append(candidates[i])
31            dfs(path, i+1, residue)
32            path.pop()
33
34    dfs(path, 0, target)
35    return res

```

子集:

78. 子集

难度 中等 705 收藏 评论 举报

给定一组不含重复元素的整数数组 *nums*，返回该数组所有可能的子集（幂集）。

说明：解集不能包含重复的子集。

示例：

```
输入：nums = [1,2,3]
输出：
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

求解方法，三种：

第一种：库函数 `itertools.combinations(nums, i)`：返回值是，*nums*中，长度为 *i*的子序列

```
1 def subsets(self, nums):
2     res = []
3     for i in range(len(nums) + 1):
4         # 返回nums中，长度为 i的所有子序列
5         for tmp in itertools.combinations(nums, i):
6             res.append(tmp)
7     return res
```

第二种：迭代

```
1 def subsets(self, nums):
2     res = [[]]
3     for i in nums:
4         res = res + [[i] + num for num in res]
5     return res
```

第三章：递归/回溯

```
1 def subsets(self, nums):
2
3     #递归法：
4     # 求长度为k的子集
5     def backtrack(first = 0, curr = []):
6         # if the combination is done
7         if len(curr) == k:
8             output.append(curr[:])
9         for i in range(first, n):
10            # add nums[i] into the current combination
11            curr.append(nums[i])
12            # use next integers to complete the combination
```

```

13         backtrack(i + 1, curr)
14         # backtrack
15         curr.pop()
16
17     output = []
18     n = len(nums)
19     # 遍历 k的所有可能
20     for k in range(n + 1):
21         backtrack()
22     return output

```

90 子集II

给定一个可能包含重复元素的整数数组 *nums*，返回该数组所有可能的子集（幂集）。

说明：解集不能包含重复的子集。

示例：

```

输入: [1,2,2]
输出:
[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]

```

最简单的想法就是利用子集，在结果中，把重复元素去掉

```

1 def subsetsWithDup(self, nums):
2     """
3     :type nums: List[int]
4     :rtype: List[List[int]]
5     """
6     if not nums:
7         return []
8     res = []
9     n = len(nums)
10    nums.sort()
11    # 求长度为k的子集
12    def backtrack(first = 0, curr = []):
13        # if the combination is done
14        if len(curr) == k:
15            # 去重
16            if curr not in output:
17                output.append(curr[:])
18        for i in range(first, n):
19            # add nums[i] into the current combination

```



```

20         curr.append(nums[i])
21         # use next integers to complete the combination
22         backtrack(i + 1, curr)
23         # backtrack
24         curr.pop()
25
26     output = []
27     n = len(nums)
28     # 遍历 k的所有可能
29     for k in range(n + 1):
30         backtrack()
31     return output

```

递归法二：

```

1  def subsetsWithDup(self, nums):
2      if not nums:
3          return []
4      res = []
5      n = len(nums)
6      nums.sort()
7      def helper(i, tmp):
8          if tmp not in res:
9              res.append(tmp)
10             for j in range(i, n):
11                 helper(j + 1, tmp + [nums[j]] )
12         helper(0, [])
13     return res

```