0.回溯算法总结.md

回溯是一种搜索方式，主要用来寻找所有解的可能，回溯都依赖递归实现，属于递归的一个方向。一般抽象为多叉树的深度搜索问题，候选集的大小是树的宽度，target是树的深度。空间复杂度一般是深度搜索的深度。时间复杂度不定，不过回溯并不高效

- 求所有组合：有重复数组，无重复数组（可/不可重复取）
    - 17 电话号码的字母组合：无重复，一次使用
    - 39 组合总和：无重复，反复使用，sort剪枝
    - 40 组合总和 II：有重复，sort，判断去重
    - 77 组合：无重复，一次使用
    - 216 组合总和 III：求和为n，注意遍历右边界要包括n
    - 22 括号生成：left+right counter判断是否生成右括号
    - 空间复杂度O(n)：递归的深度
    - 时间复杂度O(2^n*n): 虽然题目各不相同，基本不存在重复遍历所以是组合数量
- 求所有排列：数组有/无重复
    - 46 全排列: 无重复，递归包含自己
    - 47 全排列 II：有重复，sort，判断去重
    - 空间复杂度O(n)：递归的深度
    - 时间复杂度O(n!)：排列的数量
- 分割问题：把字符串/数组按要求分割
    - 131 分割回文串：在组合的基础上，加一步是否回文的判断
    - 93 复原 IP 地址：在组合的基础上，加一步是否满足IP的判断
    - 空间复杂度O(n)
    - 时间复杂度
- 求子集
    - 78 子集：无重复，在组合的基础上，每一步都append result
    - 90 子集 II：有重复，同上
    - 空间复杂度O(n)：和组合相同
    - 时间复杂度O(2^n*n): 和组合相同
- 求子序列
    - 491 递增子序列：注意是按原顺序递增，因为不能sort，所以只能通过set来判断解是否重复
    - 空间复杂度O(n)
    - 时间复杂度
- 其他
    - 79 单词搜索：回溯中异类不返回集合返回True/False，每一步回溯需要存储状态
- 回溯问题模版
    - 停止条件
    - 递归方式
    - 递归入参
    - 优化：剪枝（如果无序需要先sort）

131. 分割回文串.md

给你一个字符串 s，请你将 s 分割成一些子串，使每个子串都是 回文串 。返回 s 所有可能的分割方案。

回文串 是正着读和反着读都一样的字符串。

示例 1：

输入：s = "aab"
输出：[["a","a","b"],["aa","b"]]

示例 2：

输入：s = "a"
输出：[["a"]]

提示：

```
1 <= s.length <= 16
s 仅由小写英文字母组成
```

```python
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        ans = []

        def checker(s):
            l = len(s)
            for i in range(l//2):
                if s[i]!=s[l-1-i]:
                    return False
            return True

        def dfs(s, path):
            if not s:
                ans.append(path)
                return
            for i in range(len(s)):
                if checker(s[:i+1]):
                    dfs(s[(i+1):], path+[s[:(i+1)]])
        dfs(s, [])
        return ans
```

Tips

1. 因为不能反复切割，所以和无重复的从候选里取元素是一样的。只需要多加一步检查是否是回文，如果是加入当前path，否则直接跳过

17. 电话号码的字母组合.md

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。答案可以按 任意顺序 返回。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

示例 1：

输入：digits = "23"
输出：["ad","ae","af","bd","be","bf","cd","ce","cf"]

示例 2：

输入：digits = ""
输出：[]

示例 3：

输入：digits = "2"
输出：["a","b","c"]

提示：

```
0 <= digits.length <= 4
digits[i] 是范围 ['2', '9'] 的一个数字。
```

```python
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        if not digits:
            return list()

        phoneMap = {
            "2": "abc",
            "3": "def",
            "4": "ghi",
            "5": "jkl",
            "6": "mno",
            "7": "pqrs",
            "8": "tuv",
            "9": "wxyz",
        }
        result = []
        def dfs(digits, cur):
```

```
            if not digits:
                result.append(cur)
                return
            for s in phoneMap[digits[0]]:
                dfs(digits[1:], cur+s)
            return
        dfs(digits, '')
        return result
```

Tips

回溯算法

1. 停止条件是digits遍历完
2. 每一层都是遍历当前第一数字的所有字母，然后递归下一个数字
3. 递归入参就是未遍历的digits，已遍历的字母
4.

216. 组合总和 III.md

找出所有相加之和为 n 的 k 个数的组合。组合中只允许含有 1 - 9 的正整数，并且每种组合中不存在重复的数字。

说明：

> 所有数字都是正整数。
> 解集不能包含重复的组合。

示例 1:

输入: k = 3, n = 7
输出: [[1,2,4]]

示例 2:

输入: k = 3, n = 9
输出: [[1,2,6], [1,3,5], [2,3,4]]

```python
class Solution:
    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        ans = []
        def dfs(start_index, total, path):
            if len(path) == k:
                if total == 0:
                    ans.append(path)
                return
            for i in range(start_index, min(total+1,10)):
                dfs(i+1, total-i, path+[i])

        dfs(1, n, [])
        return ans
```

Tips

1. 其实是组合求和里最简单的一个，候选有序，无重复取。
2. 停止条件：len(path) =K
3. 每一层递归遍历[start_idnex, total]

22. 括号生成.md

数字 n 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 有效的 括号组合。

有效括号组合需满足：左括号必须以正确的顺序闭合。

示例 1：

输入：n = 3
输出：["((()))","(()())","(())()","()(())","()()()"]

示例 2：

输入：n = 1
输出：["()"]

提示：

```
1 <= n <= 8
```

```python
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        result = []
        def dfs(left, right, cur):
            if (left==0) and (right==0):
                result.append(cur)
                return
            if left>0:
                dfs(left-1, right, cur+'(')
            if right>left:
                dfs(left, right-1,cur+')')
        dfs(n,n, '')
        return result
```

Tips

回溯算法，只不过左右括号的条件不对称，左边优先级更高，右边要根据当前已有左边括号的数量而定

1. 括号是否valid的判断是任意i，[:i]之内的左括号数量>=右括号数量
2. 停止条件就是用完3个左+右括号，每一步都是优先加入左括号，以及当右<左的情况下加入左括号

39. 组合总和.md

给定一个无重复元素的正整数数组 candidates 和一个正整数 target ，找出 candidates 中所有可以使数字和为目标数 target 的唯一组合。

candidates 中的数字可以无限制重复被选取。如果至少一个所选数字数量不同，则两种组合是唯一的。

对于给定的输入，保证和为 target 的唯一组合数少于 150 个。

示例 1：

输入: candidates = [2,3,6,7], target = 7
输出: [[7],[2,2,3]]

示例 2：

输入: candidates = [2,3,5], target = 8
输出: [[2,2,2,2],[2,3,3],[3,5]]

示例 3：

输入: candidates = [2], target = 1
输出: []

示例 4：

输入: candidates = [1], target = 1
输出: [[1]]

示例 5：

输入: candidates = [1], target = 2
输出: [[1,1]]

提示：

```
1 <= candidates.length <= 30
1 <= candidates[i] <= 200
candidate 中的每个元素都是独一无二的。
1 <= target <= 500
```

1. 无剪枝

```python
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        ans = []
        def helper(candidate, path, target):
            if target<0:
                return
            if target==0:
                ans.append(path)
            if not candidate:
                return

            for i, c in enumerate(candidate):
                helper(candidate[i:], path+[c], target-c)

        helper(candidates,[], target)
        return ans
```

2. 剪枝

```python
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        ans = []
        def helper(candidate, path, target):
            if not candidate:
                return
            if target<0:
                return
            if target==0:
                ans.append(path)
```

```
            for i, c in enumerate(candidate):
                res = target-c
                if res<0:
                    break
                helper(candidate[i:], path+[c], target-c)

        candidates.sort()
        helper(candidates,[], target)
        return ans
```

Tips

1. 回溯算法中，输入无重复，每个元素可以重复使用，输出是组合。深度搜索时每次都保留[i:]的元素向前递归，这样当前元素在下一层依旧可以使用，又避免了（a,b）,(b,a)这两种解出现的可能
2. 求和问题的剪枝，一般是对candidate进行排序，res>target就停止
3. 时间复杂度: 所有可行解的长度之和
4. 空间复杂度=栈的深度，最差是O（target）

40. 组合总和 II.md

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。

注意：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,
输出:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]

示例 2:

输入: candidates = [2,5,2,1,2], target = 5,
输出:
[
[1,2,2],
[5]
]

提示:

```
1 <= candidates.length <= 100
1 <= candidates[i] <= 50
1 <= target <= 30
```

```python
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        ans = []
        def dfs(cand, path, target):
            if target< 0:
                return
            if target ==0:
                ans.append(path)
                return
            if not cand:
                return
            for i,c in enumerate(cand):
                if i > 0 and c == cand[i-1]:
                    continue
                res = target -c
                if res <0 :
                    break
                dfs(cand[(i+1):], path+[c], target-c)
        candidates.sort()
        dfs(candidates,[], target)
        return ans
```

Tips

和39题的差异是数组里面会有重复，每个数字只允许取一次，修改方案分别是

1. 只允许取一次：dfs下一层从[i+1:]开始，而不是从[i]开始
2. 为了避免重复组合的产生，需要判断当前元素和上一个元素是否重合

46. 全排列.md

给定一个不含重复数字的数组 nums ，返回其 所有可能的全排列 。你可以 按任意顺序 返回答案。

示例 1：

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

示例 2：

输入：nums = [0,1]
输出：[[0,1],[1,0]]

示例 3：

输入：nums = [1]
输出：[[1]]

提示：

```
1 <= nums.length <= 6
-10 <= nums[i] <= 10
nums 中的所有整数 互不相同
```

```python
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        ans = []
        def dfs(candidate, path):
            if not candidate:
                ans.append(path)
                return
            for i,c in enumerate(candidate):
                dfs(candidate[:i]+candidate[(i+1):], path+[c])
        dfs(nums, [] )
        return ans
```

Tips

1. 和39/40组合总和的区别在于，这里返回的是排列而非组合，输入和39题一样数字是无重复的
2. 调整：每次遍历时只剔除当前元素candidate[:i]+candidate[(i+1):] ，而非向前遍历

47. 全排列 II.md

给定一个可包含重复数字的序列 nums ， 按任意顺序 返回所有不重复的全排列。

示例 1：

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

示例 2：

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

提示：

```
1 <= nums.length <= 8
-10 <= nums[i] <= 10
```

```python
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        ans = []
        def dfs(candidate, path):
            if not candidate:
                ans.append(path)

            for i,c in enumerate(candidate):
                if i>0 and candidate[i-1]==c:
                    continue
                dfs(candidate[:i]+ candidate[(i+1):], path+[c])
        nums.sort()
        dfs(nums,[])
        return ans
```

Tips

和46的区别，输入是有重复的list，解决方案和40题一样，我们先对nums进行排序，但发现cand[i]==cand[i-1]时跳过当前元素

491. 递增子序列.md

给你一个整数数组 nums ，找出并返回所有该数组中不同的递增子序列，递增子序列中 至少有两个元素 。你可以按 任意顺序 返回答案。

数组中可能含有重复元素，如出现两个整数相等，也可以视作递增序列的一种特殊情况。

示例 1：

输入：nums = [4,6,7,7]
输出：[[4,6],[4,6,7],[4,6,7,7],[4,7],[4,7,7],[6,7],[6,7,7],[7,7]]

示例 2：

输入：nums = [4,4,3,2,1]
输出：[[4,4]]

提示：

```
1 <= nums.length <= 15
-100 <= nums[i] <= 100
```

```python
class Solution:
    def findSubsequences(self, nums: List[int]) -> List[List[int]]:
        ans = []
        def dfs(nums, path):
            if len(path)>=2:
                ans.append(path)
            if not nums:
                return

            visited = set()
            for i,n in enumerate(nums):
                if n in visited:
                    continue
                if not path or n>= path[-1]:
                    visited.add(n)
                    dfs(nums[(i+1):], path+[n])
        dfs(nums,[])
        return ans
```

Tips

1. 审题！！！！一上来我就把数组給sort了然后按照47题重复数组的路子来做。但是这里是自增（按照输入顺序递增）,所以这里跳弓重复元素的方式是在每一层递归里面用hash来保存已经遍历过的元素，避免重复遍历
2. 注意这里不需要在停止时再append，需要在len>=2之后每一步都append
3. 时间复杂度是O（n^2）因为不管咋实现本质都是判断每个元素是否加入的问题
4. 空间复杂度是O(n)=栈深度

77. 组合.md

给定两个整数 n 和 k，返回范围 [1, n] 中所有可能的 k 个数的组合。

你可以按 任何顺序 返回答案。

示例 1：

输入：n = 4, k = 2
输出：
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]

示例 2：

输入：n = 1, k = 1
输出：[[1]]

提示：

```
1 <= n <= 20
1 <= k <= n
```

1. 无剪枝回溯

```python
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        ans = []
        def dfs(start_index, path):
            if len(path) == k:
                ans.append(path)
                return
            for i in range(start_index, n+1):
                dfs(i+1, path+[i])
        dfs(1, [])
        return ans
```

Tips

1. 停止条件就是path长度=组合长度
2. 每一步都从当前index向后遍历每一个值加入到当前path里面，继续递归
3. 因为是组合数，所以需要向后遍历保证每种组合只出现一次，所以递归入参之需要传入起始参数和已经加入的path

2. 剪枝：对于从当前节点遍历不够K的path不去遍历就好，所以每一步的遍历end=n+2-k+len(path)

```python
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        ans = []
        def dfs(start_index, path):
            if len(path) == k:
                ans.append(path)
                return
            for i in range(start_index, n+2-k+len(path)):
                dfs(i+1, path+[i])
        dfs(1, [])
        return ans
```

78. 子集.md

给你一个整数数组 nums ， 数组中的元素 互不相同 。返回该数组所有可能的子集（幂集）。

解集 不能 包含重复的子集。你可以按 任意顺序 返回解集。

示例 1：

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

示例 2：

输入：nums = [0]
输出：[[],[0]]

提示：

```
1 <= nums.length <= 10
-10 <= nums[i] <= 10
nums 中的所有元素 互不相同
```

```python
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        ans = []
        def dfs(cand, path):
            ans.append(path)
            if not cand:
                return
            for i,c in enumerate(cand):
                dfs(cand[(i+1):], path+[c])
        dfs(nums, [])
        return ans
```

Tips

依回溯

返回时组合而非排序。不过这踢的特殊事没有停止条件，递归的每一步哎判断停止之前都要把结果写入

79. 单词搜索.md

给定一个 m x n 二维字符网格 board 和一个字符串单词 word 。如果 word 存在于网格中，返回 true ；否则，返回 false 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

示例 1：

输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
输出：true

示例 2：

输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"
输出：true

示例 3：

输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"
输出：false

提示：

```
m == board.length
n = board[i].length
1 <= m, n <= 6
1 <= word.length <= 15
board 和 word 仅由大小写英文字母组成
```

```python
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        direction = [(0, 1), (1, 0), (-1, 0), (0, -1)]
        visited = set()
        nrow = len(board)
        ncol = len(board[0])

        def check(i,j, s):
            if board[i][j] !=s[0]:
                return False
            if len(s)==1:
                return True
            visited.add((i,j))
            result = False
            for d in direction:
                row = i+d[0]
                col = j+d[1]
                if row>=0 and col>=0 and row<nrow and col<ncol:
                    if (row,col) not in visited:
                        if check(row, col, s[1:]):
                            result=True
                            break
            visited.remove((i,j))
            return result

        for i in range(nrow):
            for j in range(ncol):
                if check(i, j, word):
                    return True

        return False
```

Tips

1. 对每个位置进行回溯
2. 每次回溯只要存在True/False即返回

进阶：你可以使用搜索剪枝的技术来优化解决方案，使其在 board 更大的情况下可以更快解决问题?

Tips进阶的解决方案就是用额外的visited set来记录每个位置回溯中已经搜索过的位置，注意是每次回溯而不是全局搜索，因为只有单次回溯的位置不需要重复遍历

90. 子集 II.md

给你一个整数数组 nums ，其中可能包含重复元素，请你返回该数组所有可能的子集（幂集）。

解集 不能 包含重复的子集。返回的解集中，子集可以按 任意顺序 排列。

示例 1：

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

示例 2：

输入：nums = [0]
输出：[[],[0]]

提示：

```
1 <= nums.length <= 10
-10 <= nums[i] <= 10
```

```python
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        ans = []
        def dfs(cand, path):
            ans.append(path)
            if not cand:
                return

            for i,c in enumerate(cand):
                if i>0 and (cand[i-1]==c):
                    continue
                dfs(cand[(i+1):], path+[c])
        nums.sort()
        dfs(nums, [])
        return ans
```

Tips

就在子集1上多加两步

1. sort nums
2. 每一步判断当前元素是否为重复如果是跳过

93. 复原 IP 地址.md

给定一个只包含数字的字符串，用以表示一个 IP 地址，返回所有可能从 s 获得的 有效 IP 地址 。你可以按任何顺序返回答案。

有效 IP 地址 正好由四个整数（每个整数位于 0 到 255 之间组成，且不能含有前导 0），整数之间用 '.' 分隔。

例如："0.1.2.201" 和 "192.168.1.1" 是 有效 IP 地址，但是 "0.011.255.245"、"192.168.1.312" 和 "192.168@1.1" 是 无效 IP 地址。

示例 1：

输入：s = "25525511135"
输出：["255.255.11.135","255.255.111.35"]

示例 2：

输入：s = "0000"
输出：["0.0.0.0"]

示例 3：

输入：s = "1111"
输出：["1.1.1.1"]

示例 4：

输入：s = "010010"
输出：["0.10.0.10","0.100.1.0"]

示例 5：

输入：s = "101023"
输出：["1.0.10.23","1.0.102.3","10.1.0.23","10.10.2.3","101.0.2.3"]

提示：

```
0 <= s.length <= 3000
s 仅由数字组成
```

```python
class Solution:
    def restoreIpAddresses(self, s: str) -> List[str]:
        result = []
        if len(s)<4:
            return result
        def helper(cur_s, ip_list):
            #停止条件
            if not cur_s:
                if len(ip_list)==4:
                    result.append('.'.join(ip_list))
                    return
                else:
                    return
            if len(ip_list)==4:
                return
            helper(cur_s[1:], ip_list +[cur_s[0]])
            if cur_s[0]!='0':
                if len(cur_s)>=2:
                    helper(cur_s[2:], ip_list + [cur_s[:2]])
                if (len(cur_s)>=3) and (int(cur_s[:3])<=255):
                    helper(cur_s[3:], ip_list+[cur_s[:3]])
        helper(s,[])
        return result
```

Tips

回溯

1. 停止条件：cur_s已经遍历完，或者已经分割出4个整数，只有当同时满足条件的时候把结果加入ans
2. 递归：这里因为有各种限制条件所以不能简单的for i in range（4）的去遍历，2/3个元素需要分别处理