▶

In [ ]:

```python
# 77 组合
def combine(n, k):
    # [1,2,3,...,n]取k个数
    res = []
    path = []
    def backtrack(n, k, startIndex):
        # startIndex控制了输入（且只往后取，和全排列有所不同）= pop
        # 终止条件
        if len(path) == k:
            res.append(path[:])
            return
        for i in range(startIndex, n-(k-len(path))+2):  # 层的可选项
            # n-(k-len(path))+2 举例子，n=4,k=3,path=[],最多是[2,3,4]，取2，而4-(3-0)+2=3,但是不
            # n是从1开始取，取相应的值时必须+1
            path.append(i)  # 选择
            backtrack(n, k, i+1) # n、k、i+1 共同控制了层的大小
            path.pop() # 回退，但是可选项不需要补全，因为不能往前面取了
    backtrack(n, k, 1)
    return res

# 216 组合求和III，求和问题I
def combinationSum3(k, n) :
    # k 个数，和为n，取值[1,2,3,...,9]
    res = []
    path = []
    def backtrack(targetSum, Sum, k, startIndex):
        if Sum > n : return
        if len(path) == k and targetSum == Sum:
                return res.append(path[:])
        for i in range(startIndex, 9-(k-len(path))+2):
            Sum += i
            path.append(i)
            backtrack(targetSum, Sum, k, i+1)
            Sum -= i
            path.pop()
    backtrack(n, 0, k, 1)
    return res


# 17 电话号码的字母组合
def letterCombinations(digits):
    result = []  # 可全局调用
    self.s = ''  # 不可全局调用
    letterMap = ["","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"]
    if len(digits) == 0: return []
    def backtrack(digits, index):
        # index决定在哪一层,表示取digits的第index位开始取
        if index == len(digits): return result.append(self.s)
        digit = int(digits[index])
        letters = letterMap[digit]
        for i in range(len(letters)):
            # 排列问题不需要修补letters
            self.s += letters[i]
            backtrack(digits, index+1)
            self.s  = self.s[:-1]
    backtrack(digits, 0)
    return result
```

```python
58
59  # 46 全排列(没有重复数字)
60  def permute(self, nums: List[int]) -> List[List[int]]:
61      if nums == []:return []
62      results = []
63      res = []
64      def backtrack(num):
65          if num == []:
66              results.append(res[:])    # 非常重要
67          for i in range(len(num)):
68              c = num.pop(i)
69              res.append(c)
70              backtrack(num)
71              num.insert(i,c)
72              res.pop()
73      backtrack(nums)
74      return results
75
76  # 47 全排II(有重复数字)
77  def permuteUnique(nums):
78      if nums == []: return []
79      results = []
80      res = []
81      def backtrack(num):
82          if num == [] and res[:] not in results: # 区别在去重
83              results.append(res[:])     # 深拷贝，需要复制
84          for i in range(len(num)):
85              c = num.pop(i)
86              res.append(c)
87              backtrack(num)
88              num.insert(i,c)
89              res.pop()
90      backtrack(nums)
91      return results
92
93  # 字符排列
94  def Permutation(ss):
95      if ss == '': return ''
96      result = []
97      res = []
98      def backtrack(string):
99          string = list(string)
100         if string == []:
101             result.append(''.join(res)) # 已经不可修改了
102         for i in range(len(string)):
103             c = string.pop(i)
104             if c:
105                 res.append(c)
106                 backtrack(string)
107                 string.insert(i,c)
108                 res.pop()
109     backtrack(ss)
110     return sorted(list(set(result)))
111
112
113 # 39 组合总和II
114 def combinationSum(candidates, target):
115     res = []
116     path = []
117     def backtrack(target, Sum, startIndex, candidates):
118         if Sum == target: return res.append(path[:])
```

```python
119            for i in range(startIndex, len(candidates)):
120                if Sum + candidates[i]> target: return   # 经过排序后可以用，直接跳出
121                Sum += candidates[i]
122                path.append(candidates[i])
123                backtrack(target, Sum, i, candidates) # i表示能 重复使用自己
124                Sum -= candidates[i]
125                path.pop()
126        candidates = sorted(candidates)
127        backtrack(target, 0, 0, candidates)
128        return res
129
130 # 40 组合求和III
131 def combinationSum2(candidates, target)
132        res = []
133        path  = []
134        def backtrack(target, Sum, startIndex, candidates):
135            if Sum == target: return res.append(path[:])
136            for i in range(startIndex, len(candidates)):
137                if Sum + candidates[i] > target: return
138                if i > startIndex and candidates[i] == candidates[i-1]: continue   # 加了这样行去
139                Sum += candidates[i]
140                path.append(candidates[i])
141                backtrack(target, Sum, i+1, candidates) # i+1 表示当前字符能否重复使用自己，i+1表示不育
142                Sum -= candidates[i]
143                path.pop()
144        candidates = sorted(candidates)
145        backtrack(target, 0, 0, candidates)
146        return res
147
148 # 131 分割回文字符
149 def partition(s):
150        res = []
151        path = []
152        def backtrack(s, startIndex):
153            if startIndex == len(s) : return res.append(path[:])
154            for i in range(startIndex, len(s)):
155                p = s[startIndex:i+1]
156                if p == p[::-1]:     # 是回文字符，才递归，不是的直接扔掉
157                    path.append(p)
158                    backtrack(s, i+1)
159                    path.pop()
160        backtrack(s, 0)
161        return res
162
163 # 93 复原IP
164  def restoreIpAddresses(s):
165        res = []
166        path = []
167        def backtrack(s, level):
168            if level == 5 and s == '' and '.'.join(path) not in res:
169                return res.append('.'.join(path))
170            if level ==5 or s == '': return
171            for i in range(1, 4):
172                x = s[:i]
173                if int(x) < 256 and (x =='0' or x[0] !='0'):
174                    path.append(x)
175                    backtrack(s[i:], level+1)
176                    path.pop()
177        backtrack(s, 1)
178        return res
179
```

```python
# 78 子集I
def subsets(nums):
    res = []
    path = []
    def backtrack(startIndex, nums):
        res.append(path[:])   # [:] ,第一个为空，加入，来了先加入
        if startIndex >= len(nums): return
        for i in range(startIndex, len(nums)):
            path.append(nums[i])
            backtrack(i+1, nums)
            path.pop()
    backtrack(0, nums)
    return res

# 90 子集II
def subsetsWithDup(nums):
    res = []
    path = []
    def backtrack(startIndex, nums):
        res.append(path[:])
        if startIndex == len(nums): return
        for i in range(startIndex, len(nums)):
            if i > startIndex and nums[i] == nums[i-1]: continue #我们要对同一树层使用过的元
            path.append(nums[i])
            backtrack(i+1, nums)
            path.pop()
    nums = sorted(nums)
    backtrack(0, nums)
    return res

# 491 递增子序列
def findSubsequences(nums):
    # 给的例子是个坑，不能对序列进行排序，找这个顺序下的最长子序列
    res = []
    path = []
    def backtrack(startIndex, nums):
        repeat = [] # 同一层下不重复出现，树枝中可重复
        if len(path) > 1: res.append(path[:])
        for i in range(startIndex, len(nums)):
            if nums[i] in repeat: continue  # 往后走
            if len(path) > 0 and nums[i] < path[-1]: continue  # 往后走
            repeat.append(nums[i])
            path.append(nums[i])
            backtrack(i+1, nums)
            path.pop()
    backtrack(0, nums)
    return res

# 51 N皇后问题
def solveNQueens(n):
    if not n: return []
    board = [['.']*n  for i in range(n)]  # 构建棋盘
    res = []
    def isVaild(board, row, col):
        # 简单
        # 判断同一列是否冲突
        for i in range(len(board)):
            if board[i][col] == 'Q': return False
        # 判断同一左斜线冲突
        i = row - 1
        j = col - 1
```

```
241            while i >=0 and j >= 0:
242                if board[i][j] == 'Q': return False
243                i -= 1
244                j -= 1
245            # 判断同一右斜线冲突
246            i = row - 1
247            j = col + 1
248            while i >=0 and j < len(board):
249                if board[i][j] == 'Q': return False
250                i -= 1
251                j += 1
252            return True
253        def backtrack(board, row, n):
254            if row == len(board):  res.append([''.join(i) for i in board]) # 特殊格式
255            for col in range(n):
256                if not isVaild(board, row, col): continue
257                board[row][col] = 'Q'
258                backtrack(board, row+1, n)
259                board[row][col] = '.'
260        backtrack(board, 0, n)
261        return res
262
263    # 数独判断
264    def solveSudoku(board):
265        """
266        Do not return anything, modify board in-place instead. 只做方法修改即可
267        """
268        def isValid(row, col, val, board):
269            for i in range(9): # 判断同行
270                if board[row][i] == str(val): return False
271            for i in range(9): # 判断 同列
272                if board[i][col] == str(val): return False
273            startRow = (row // 3) *3
274            startCol = (col // 3) *3
275            for i in range(startRow, startRow+3): # 判断同个9*9
276                for j in range(startCol, startCol+3):
277                    if board[i][j] == str(val):return False
278            return True
279
280        def backtrack(board):
281            # 直接修改board
282            for i in range(len(board)):
283                for j in range(len(board[0])):
284                    if board[i][j] != '.': continue
285                    for num in range(1, 10):
286                        if isValid(i, j, num, board):
287                            board[i][j] = str(num)
288                            if backtrack(board) : return True
289                            board[i][j] = '.' # 不填num, 换一个数
290                    return False
291            return True
292        backtrack(board)
293
294    # 22 括号生成
295    def generateParenthesis(n):
296        if n == 0: return []
297        res = []
298        self.path = ''   # 字符串只能这样定义全局变量
299        def backtrack(left,right):
300            # left 左边括号数， right 右边括号数
301            if left < 0 or right < 0: return
```

```
302        if left > right: return
303        if left == 0 and right == 0: return res.append(self.path)
304        self.path += '('
305        backtrack(left-1, right)
306        self.path = self.path[:-1]
307        self.path += ')'
308        backtrack(left, right-1)
309        self.path = self.path[:-1]
310    backtrack(n, n)
311    return res
```