



In [ ]:

```

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

20
21 # 216 组合求和III，求和问题I
22 def combinationSum3(k, n):
23     # k 个数，和为n，取值[1, 2, 3, ..., 9]
24     res = []
25     path = []
26     def backtrack(targetSum, Sum, k, startIndex):
27         if Sum > n: return
28         if len(path) == k and targetSum == Sum:
29             return res.append(path[:])
30         for i in range(startIndex, 9 - (k - len(path)) + 2):
31             Sum += i
32             path.append(i)
33             backtrack(targetSum, Sum, k, i + 1)
34             Sum -= i
35             path.pop()
36         backtrack(n, 0, k, 1)
37     return res

38
39
40 # 17 电话号码的字母组合
41 def letterCombinations(digits):
42     result = [] # 可全局调用
43     self.s = '' # 不可全局调用
44     letterMap = ['', '', 'abc', 'def', 'ghi', 'jkl', 'mno', 'pqrs', 'tuv', 'wxyz']
45     if len(digits) == 0: return []
46     def backtrack(digits, index):
47         # index决定在哪一层，表示取digits的第index位开始取
48         if index == len(digits): return result.append(self.s)
49         digit = int(digits[index])
50         letters = letterMap[digit]
51         for i in range(len(letters)):
52             # 排列问题不需要修补letters
53             self.s += letters[i]
54             backtrack(digits, index + 1)
55             self.s = self.s[:-1]
56         backtrack(digits, index)
57     return result

```

此处n相当于一个可选列表，  
 n + startIndex 控制了选择列表  
 for 选择列表：  
 选一个  
 回溯  
 退选  
 递归含义：  
 1. startIndex 开始，并判断  
 长度是否满足  
 限制个数k，字从n中取  
 backtrack 控制结束参数，  
 选择列表参数  
 长为k，和为target  
 选择列表  
 数字  
 digits: 数组  
 index: 第几个数  
 做选择

```

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             return 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 # 39 组合总和II 有限数 [ ] 无重复, 但可重复使用自己。
113 def combinationSum(candidates, target):
114     res = []
115     path = []
116     def backtrack(target, Sum, startIndex, candidates):
117         if Sum == target: return res.append(path[:])
118

```

```

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 无限个数
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表示不能
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] # 选择
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('.' + s)
170         if level == 5 or s == '': return
171         for i in range(1, 4):
172             x = s[:i] # 取, 选择
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, level+1)
178     return res
179

```

```

180 # 78 子集I
181 def subsets(nums):
182     res = []
183     path = []
184     def backtrack(startIndex, nums):
185         res.append(path[:]) # [:], 第一个为空, 加入, 来了先加入
186         if startIndex >= len(nums): return
187         for i in range(startIndex, len(nums)):
188             path.append(nums[i])
189             backtrack(i+1, nums)
190             path.pop()
191     backtrack(0, nums)
192     return res
193
194 # 90 子集II
195 def subsetsWithDup(nums):
196     res = []
197     path = []
198     def backtrack(startIndex, nums):
199         res.append(path[:])
200         if startIndex == len(nums): return
201         for i in range(startIndex, len(nums)):
202             if i > startIndex and nums[i] == nums[i-1]: continue #我们要对同一树层使用过的元
203             path.append(nums[i])
204             backtrack(i+1, nums)
205             path.pop()
206     nums = sorted(nums)
207     backtrack(0, nums)
208     return res
209
210 # 491 递增子序列
211 def findSubsequences(nums):
212     # 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
213     res = []
214     path = []
215     def backtrack(startIndex, nums):
216         repeat = [] # 同一层下不重复出现, 树枝中可重复
217         if len(path) > 1: res.append(path[:])
218         for i in range(startIndex, len(nums)):
219             if nums[i] in repeat: continue # 往后走
220             if len(path) > 0 and nums[i] < path[-1]: continue # 往后走
221             repeat.append(nums[i])
222             path.append(nums[i])
223             backtrack(i+1, nums)
224             path.pop()
225     backtrack(0, nums)
226     return res
227
228 # 51 N皇后问题
229 def solveNQueens(n):
230     if not n: return []
231     board = [['.']*n for i in range(n)] # 构建棋盘
232     res = []
233     def isValid(board, row, col):
234         # 简单
235         # 判断同一列是否冲突
236         for i in range(len(board)):
237             if board[i][col] == 'Q': return False
238         # 判断同一左斜线冲突
239         i = row - 1
240         j = col - 1

```

{1}, {1, 2}, {1, 2, 3}  
不用递归的  
从原序列中取  
一行一行填充  
定义好游戏规则  
找3种不符条件

```

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 isValid(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

```

# 数独判断

```

263 def solveSudoku(board):
264     """
265     Do not return anything, modify board in-place instead. 只做方法修改即可
266     """
267     def isValid(row, col, val, board):
268         for i in range(9): # 判断同行
269             if board[row][i] == str(val): return False
270         for i in range(9): # 判断 同列
271             if board[i][col] == str(val): return False
272         startRow = (row // 3) * 3
273         startCol = (col // 3) * 3
274         for i in range(startRow, startRow+3): # 判断同个9*9
275             for j in range(startCol, startCol+3):
276                 if board[i][j] == str(val): return False
277         return True
278
279     def backtrack(board):
280         # 直接修改board
281         for i in range(len(board)):
282             for j in range(len(board[0])):
283                 if board[i][j] != '.': continue
284                 for num in range(1, 10):
285                     if isValid(i, j, num, board):
286                         board[i][j] = str(num)
287                         if backtrack(board): return True
288                         board[i][j] = '.' # 不填num, 换一个数
289             return False
290         return True
291     backtrack(board)

```

# 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

```

for

for i in '()':  
 self.path += i  
 if i == '(': backtrack(left-1, right)  
 else: backtrack(left, right-1)  
 self.path = self.path[:-1]

byteQiu