216-组合总和Ⅲ

题述

216. 组合总和 III

难度 中等 461

找出所有相加之和为 `n` 的 `k` 个数的组合，且满足下列条件：

- 只使用数字1到9
- 每个数字 **最多使用一次**

返回 *所有可能的有效组合的列表* 。该列表不能包含相同的组合两次，组合可以以任何顺序返回。

示例 1:

输入: *k* = 3, *n* = 7
输出: [[1,2,4]]
解释:
1 + 2 + 4 = 7
没有其他符合的组合了。

示例 2:

输入: *k* = 3, *n* = 9
输出: [[1,2,6], [1,3,5], [2,3,4]]
解释:
1 + 2 + 6 = 9
1 + 3 + 5 = 9
2 + 3 + 4 = 9
没有其他符合的组合了。

示例 3:

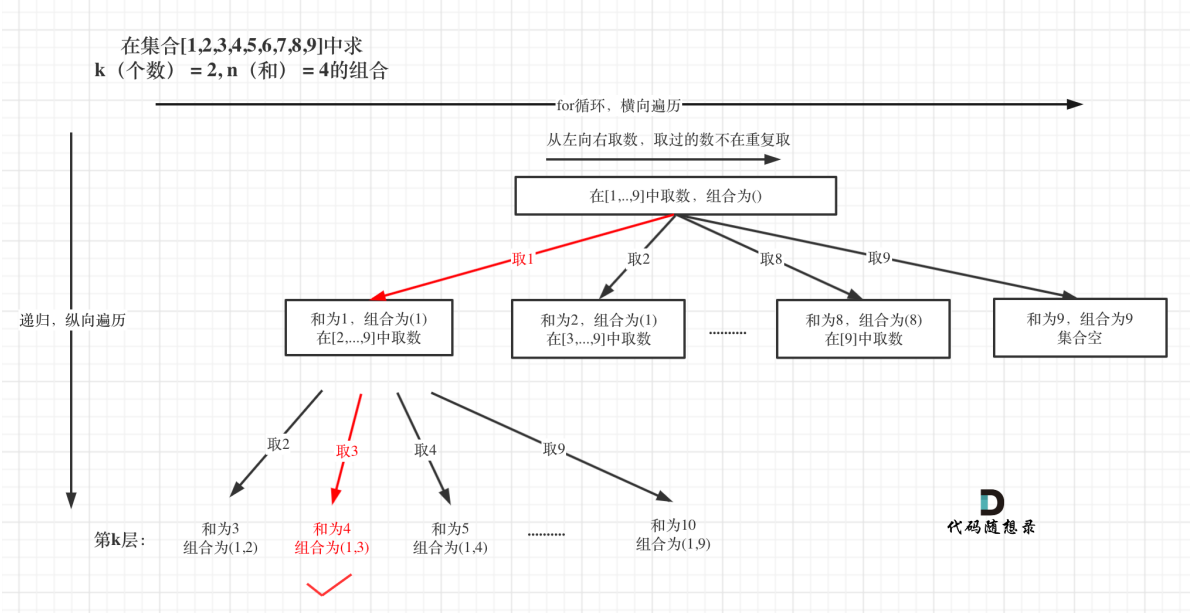
输入: k = 4, n = 1
输出: []
解释: 不存在有效的组合。
在[1,9]范围内使用4个不同的数字，我们可以得到的最小和是1+2+3+4 = 10，因为10 > 1，没有有效的组合。

思路

本题就是在[1,2,3,4,5,6,7,8,9]这个集合中找到和为n的k个数的组合。

本题k相当于了树的深度，9（因为整个集合就是9个数）就是树的宽度。

例如 k = 2，n = 4的话，就是在集合[1,2,3,4,5,6,7,8,9]中求 k（个数） = 2, n（和） = 4的组合。



回溯三板斧

确定递归函数参数

需要一维数组path来存放符合条件的结果，二维数组result来存放结果集。Python中则无需考虑这些

接下来还需要如下参数：

- targetSum（int）目标和，也就是题目中的n。
- k（int）就是题目中要求k个数的集合。
- startIndex（int）为下一层for循环搜索的起始位置。

```
def backTracking(self,k:int,n:int,startIndex:int):
```

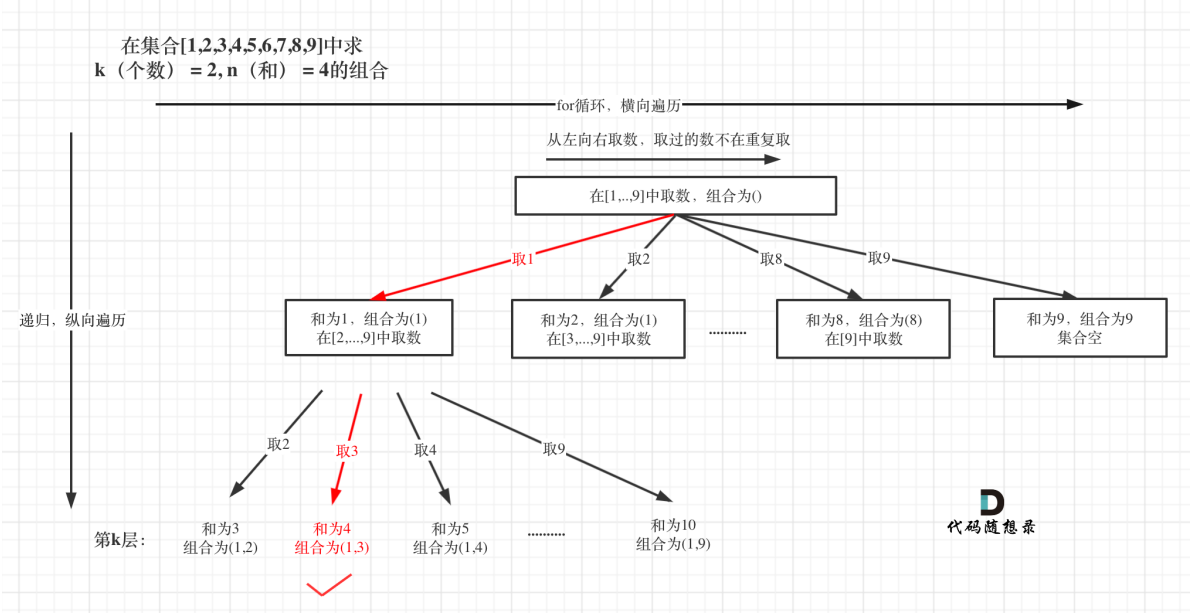
确定终止条件

当遍历到叶子节点时自然终止

```
if (path.size() == k) {
    if (sum == targetSum) result.push_back(path);
    return; // 如果path.size() == k 但sum != targetSum 直接返回
}
```

横向遍历过程

集合固定的就是9个数[1,...,9]，所以for循环固定i<=9



处理过程就是 path收集每次选取的元素，相当于树型结构里的边，sum来统计path里元素的总和。

题解

Python

```
class Solution:
    def __init__(self):
        self.result = []     #定义最终的结果数组
        self.sumNow = 0      #当前所得的和
        self.path = []       #存放符合条件的叶子节点

    def backTracking(self,k:int,n:int,startIndex:int):
        if len(self.path) == k: #终止条件 遍历到叶子节点
            if self.sumNow == n:     #如果当前所得和等于输入的n 那么满足题述条件 加入到
result中
                self.result.append(self.path[:])     #加入到result中
            return
        for i in range(startIndex,10):  #横向遍历 从startIndex 遍历到9
            self.path.append(i)
            self.sumNow += i
            self.backTracking(k,n,i+1)  #再次纵向遍历
            self.path.pop() #回溯
            self.sumNow -= i #回溯

    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        # 回溯
        self.backTracking(k,n,1)
        return self.result
```

剪枝优化

```
class Solution:
    def __init__(self):
        self.result = []     #定义最终的结果数组
        self.sumNow = 0      #当前所得的和
```

```
        self.path = []       #存放符合条件的叶子节点

    def backTracking(self,k:int,n:int,startIndex:int):
        if self.sumNow > n: #剪枝
            return
        if len(self.path) == k: #终止条件 遍历到叶子节点
            if self.sumNow == n:     #如果当前所得和等于输入的n 那么满足题述条件 加入到result中
                self.result.append(self.path[:])    #加入到result中
            return
        for i in range(startIndex,10 - (k-len(self.path))+1):  #横向遍历 从startIndex 遍历到9
            self.path.append(i)
            self.sumNow += i
            self.backTracking(k,n,i+1)  #再次纵向遍历
            self.path.pop() #回溯
            self.sumNow -= i #回溯

    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        # 回溯
        self.backTracking(k,n,1)
        return self.result
```

思考

画出树形图 然后思考遍历

剪枝优化