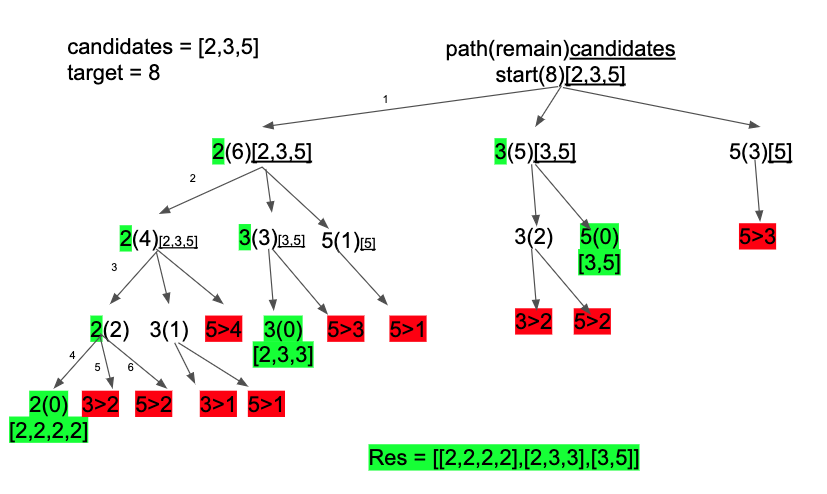
<https://leetcode.com/problems/combination-sum/discuss/815714/DFSrecursion-solution-with-illustration-to-understand-the-process>

DFS/recursion solution with illustration to understand the process

I'm using candidates[2,3,5] and target 8 as example.  
It's pretty clear with the chart below how the function dfs is called recursively and how duplications like [3,5], [5,3] is avoided.  
Some solutions sorted the candidates first, and some also have a if statement on if target <0. Both are not necessary.  
Also, I acutally found that sort the candidates may increase the number of operations. You can draw a similar chart with [5,3,2]. It has fewer operations. See codes following the chart below.  


You could also refer to my post to other similar problems to nail this type of questions once for all:  
[39. Combination Sum](https://leetcode.com/problems/combination-sum/discuss/815714/DFSrecursion-solution-with-illustration-to-understand-the-process)  
[40. Combination Sum II](https://leetcode.com/problems/combination-sum-ii/discuss/815837/Backtrackingrecursion-python-solution-with-illustration)  
[46. Permutations](https://leetcode.com/problems/permutations/discuss/816693/python-solutionor93or-utilized-the-same-logic-for-problem-39-and-40or-with-illustration)  
[93. Restore IP addresses](https://leetcode.com/problems/restore-ip-addresses/discuss/818933/python-backtracking-solution-or99or-with-illustration-and-example)

'''

def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:

res = []

self.dfs(candidates, target,[],res)

return res

def dfs(self,candidates, target, path, res):

if target==0:

res.append(path)

return

for i in range(len(candidates)):

if candidates[i]>target:

continue

self.dfs(candidates[i:],target-candidates[i],path+[candidates[i]],res)

'''