39. Combination Sum

Created	@July 17, 2020 3:04 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/combination-sum/
Importance	*****
Tag	Backtrack DFS NEET
Video	https://www.youtube.com/watch?v=lJ0qw4vr0_w&list=PL2rWx9cCzU85RX9NeRMVUV_kgI4YGKURD&index=3

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

```
Input: candidates = [2], target = 1
Output: []
```

Constraints:

- `1 <= candidates.length <= 30`
- `2 <= candidates[i] <= 40`
- All elements of `candidates` are **distinct**.
- `1 <= target <= 500`

Solution

你需要先看前文 回溯算法详解 和 回溯算法团灭子集、排列、组合问题，然后看这道题就很简单了，无非是回溯算法的运用而已。

这道题的关键在于 `candidates` 中的元素可以复用多次，体现在代码中是下面这段：

```
void backtrack(int[] candidates, int start, int target, int sum) {
    // 回溯算法框架
    for (int i = start; i < candidates.length; i++) {
        // 选择 candidates[i]
        backtrack(candidates, i, target, sum);
        // 撤销选择 candidates[i]
    }
}
// 详细解析参见：
// https://labuladong.github.io/article/?qno=39
```

对比 回溯算法团灭子集、排列、组合问题 中不能重复使用元素的标准组合问题：

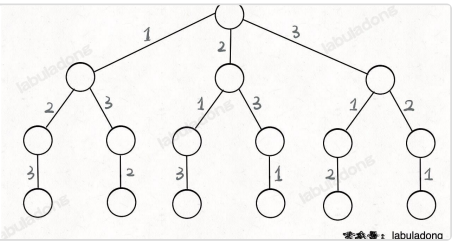
```
void backtrack(int[] candidates, int start, int target, int sum) {
    // 回溯算法框架
    for (int i = start; i < candidates.length; i++) {
        // 选择 candidates[i]
        backtrack(candidates, i + 1, target, sum);
        // 撤销选择 candidates[i]
    }
}
// 详细解析参见：
// https://labuladong.github.io/article/?qno=39
```

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        results = []
        self.dfs(candidates, target, 0, [], results)
        return results

    def dfs(self, candidates, target, start, combination, results):
        if target < 0:
            return

        if target == 0:
            return results.append(list(combination))

        for i in range(start, len(candidates)):
            combination.append(candidates[i])
            self.dfs(candidates, target-candidates[i], i, combination, results)
            combination.pop()
```

回溯算法解题套路框架

通知： 数据结构精品课 已更新到 V2.0，第 14 期打卡训练营开始报名。 读完本文，你不仅学会了算法套路，还可以顺便解决如下题目： ---- 这篇文章是很久之前的一篇 回溯算法详解 的进阶版，之前那篇不够清楚，就不必看

https://labuladong.github.io/algo/4/31/104/



回溯算法秒杀所有排列-组合-子集问题

通知： 数据结构精品课 已更新到 V2.0，第 14 期打卡训练营开始报名。 读完本文，你不仅学会了算法套路，还可以顺便解决如下题目： ---- 本文有视频版： 回溯算法秒杀所有排列/组合/子集问题 虽然排列、组合、子集系列问

https://labuladong.github.io/algo/4/31/106/