# 题目

给你一个 无重复元素 的整数数组 candidates 和一个目标整数 target ，找出 candidates 中可以使数字和为目标数 target 的 所有 不同组合 ，并以列表形式返回。你可以按 任意顺序 返回这些组合。

candidates 中的 同一个 数字可以 无限制重复被选取 。如果至少一个数字的被选数量不同，则两种组合是不同的。

对于给定的输入，保证和为 target 的不同组合数少于 150 个。

示例 1：

输入：candidates = [2,3,6,7], target = 7

输出：[[2,2,3],[7]]

解释：

2 和 3 可以形成一组候选，2 + 2 + 3 = 7 。注意 2 可以使用多次。

7 也是一个候选， 7 = 7 。

仅有这两种组合。

示例 2：

输入: candidates = [2,3,5], target = 8

输出: [[2,2,2,2],[2,3,3],[3,5]]

示例 3：

输入: candidates = [2], target = 1

输出: []

提示：

1 <= candidates.length <= 30

2 <= candidates[i] <= 40

candidates 的所有元素 互不相同

1 <= target <= 40

注意：本题与主站 39 题相同：

<https://leetcode-cn.com/problems/combination-sum/>

# 分析

## 方法一：回溯法

这道题可以使用回溯法来解决。具体思路如下：

1、对 candidates 数组进行排序，以便在回溯过程中剪枝。

2、使用一个递归函数dfs，该函数用于在当前位置start开始搜索符合条件的组合。

3、在dfs函数中，先判断当前的和sum是否等于target，如果是则将当前组合加入结果中。

4、然后从start开始遍历candidates数组，对于每个数字，如果加上candidates[i]不超过target，则将candidates[i]加入当前组合，递归调用dfs，然后在退出递归时，将candidates[i]移出当前组合，继续遍历下一个数字。

5、在遍历过程中，由于candidates数组是有序的，如果加上当前数字后已经超过了target，则可以直接退出循环，因为后面的数字只会更大，不可能满足条件。

这样可以保证找到所有符合条件的组合。

代码：

class Solution {

public:

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

vector<vector<int>> result; // 存放结果的二维数组

vector<int> path; // 存放当前组合的一维数组

sort(candidates.begin(), candidates.end()); // 对 candidates 数组排序

dfs(candidates, target, 0, path, result); // 调用回溯函数

return result;

}

private:

void dfs(vector<int>& candidates, int target, int start, vector<int>& path, vector<vector<int>>& result) {

if (target == 0) { // 当前和等于目标值，将当前组合加入结果中

result.push\_back(path);

return;

}

for (int i = start; i < candidates.size(); ++i) {

if (target - candidates[i] < 0) break; // 剪枝：如果加上当前数字超过了目标值，则退出循环

path.push\_back(candidates[i]); // 将当前数字加入当前组合

dfs(candidates, target - candidates[i], i, path, result); // 递归调用，在当前数字的基础上继续搜索

path.pop\_back(); // 回溯，将当前数字移出当前组合，继续下一轮循环

}

}

};

## 方法二：递归法

class Solution {

public:

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

vector<vector<int>> result; // 存放结果的二维数组

vector<int> path; // 存放当前组合的一维数组

sort(candidates.begin(), candidates.end()); // 对 candidates 数组排序

dfs(candidates, target, 0, path, result); // 调用回溯函数

return result;

}

private:

void dfs(vector<int>& candidates, int target, int start, vector<int>& path, vector<vector<int>>& result) {

if (target == 0) { // 当前和等于目标值，将当前组合加入结果中

result.push\_back(path);

return;

}

for (int i = start; i < candidates.size(); ++i) {

if (target - candidates[i] < 0) break; // 剪枝：如果加上当前数字超过了目标值，则退出循环

path.push\_back(candidates[i]); // 将当前数字加入当前组合

dfs(candidates, target - candidates[i], i, path, result); // 递归调用，在当前数字的基础上继续搜索

path.pop\_back(); // 回溯，将当前数字移出当前组合，继续下一轮循环

}

}

};