1. 题目
2. 题干

给定一个数组 candidates 和一个目标数 target ，找出 candidates 中所有可以使数字和为 target 的组合。

candidates 中的每个数字在每个组合中只能使用一次。

说明：

所有数字（包括目标数）都是正整数。

解集不能包含重复的组合。

1. 示例

示例 1:

输入: candidates = [10,1,2,7,6,1,5], target = 8,

所求解集为:

[

[1, 7],

[1, 2, 5],

[2, 6],

[1, 1, 6]

]

示例 2:

输入: candidates = [2,5,2,1,2], target = 5,

所求解集为:

[

  [1,2,2],

  [5]

]

1. 题解
2. 思路

这种题目，一看就知道是用递归和回溯来解决的，但是，如何写递归和回溯就是一个问题了。首先，我们要明确，每一个递归函数中需要有什么？递归需要终止，我们需要设置终止的条件，其次，如果递归没有终止，我们需要继续往下递归，那么，继续向下递归就需要根据题目设置下一次递归的条件。

当无法继续递归后，即递归到头后，需要进行回溯操作，回溯即回到上一次递归的状态。

从这个题来看，递归的终止条件是找到了等于target的组合，最终target为0，我们就将该组合temp加入ans的数组中。

当递归没有终止时，我们需要进入下一个递归中，需要满足什么条件才能够进入下一个递归中呢？首先，下一个数的索引必须大于index，且小于candidates.size()；同时，target-candidates[i]>=0；并且，candidates中重复的项是不能加入到temp数组中的。由于我们在主函数中进入递归前对candidates数组进行了排序，因此若该数组汇总有重复项，那么一定是连在一起的，因此若检查到该数在candidates数组中和前一个数是相同的，那么直接continue即可，略过这个数。

这样，我们就满足了进入下一个递归的条件，首先将temp数组加上目前的数candidates[i[，然后进入下一个递归中。反正递归就是在不断地进入递归，判断是否满足递归终止条件，如果不满足，则判断是否满足进入下一个递归的条件，若不满足，说明需要进行回溯操作。

回溯操作一般是写在每个递归函数的下面，当递归函数无法继续执行下去时，就会执行回溯的语句。回溯即回到上一个递归的状态，本题中，在递归函数执行后，若无法进入下一个递归中，就执行回溯语句。本题中为了进入下一个递归，将temp添加了当前的数candidates[i]，因此，回溯就要将刚刚加入的candidates[i]从temp中清除即可。

总结：终止条件 + 下一个递归 + 回溯，就构成了一个递归函数的架构，我们每次遇到这种题目时，就依据这种架构来编写代码即可。

1. 代码实现

C++：

class Solution {

public:

    vector<vector<int>> ans;

    vector<int> temp ;

    //回溯函数

    void backtrace(vector<int>& candidates, int target, int index){

        if(target == 0){

            ans.emplace\_back(temp);

            return;

        }

        for(int i = index; i < candidates.size() && target - candidates[i] >=0; ++i){

            //判断前后值相同不选

            if( i > index && candidates[i] == candidates[i-1]){

                continue;

            }

            temp.push\_back(candidates[i]);

            backtrace(candidates, target-candidates[i], i+1);

            temp.pop\_back();

        }

    }

    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {

        //排序后

        sort(candidates.begin(), candidates.end());

        backtrace(candidates, target, 0);

        return ans;

    }

};