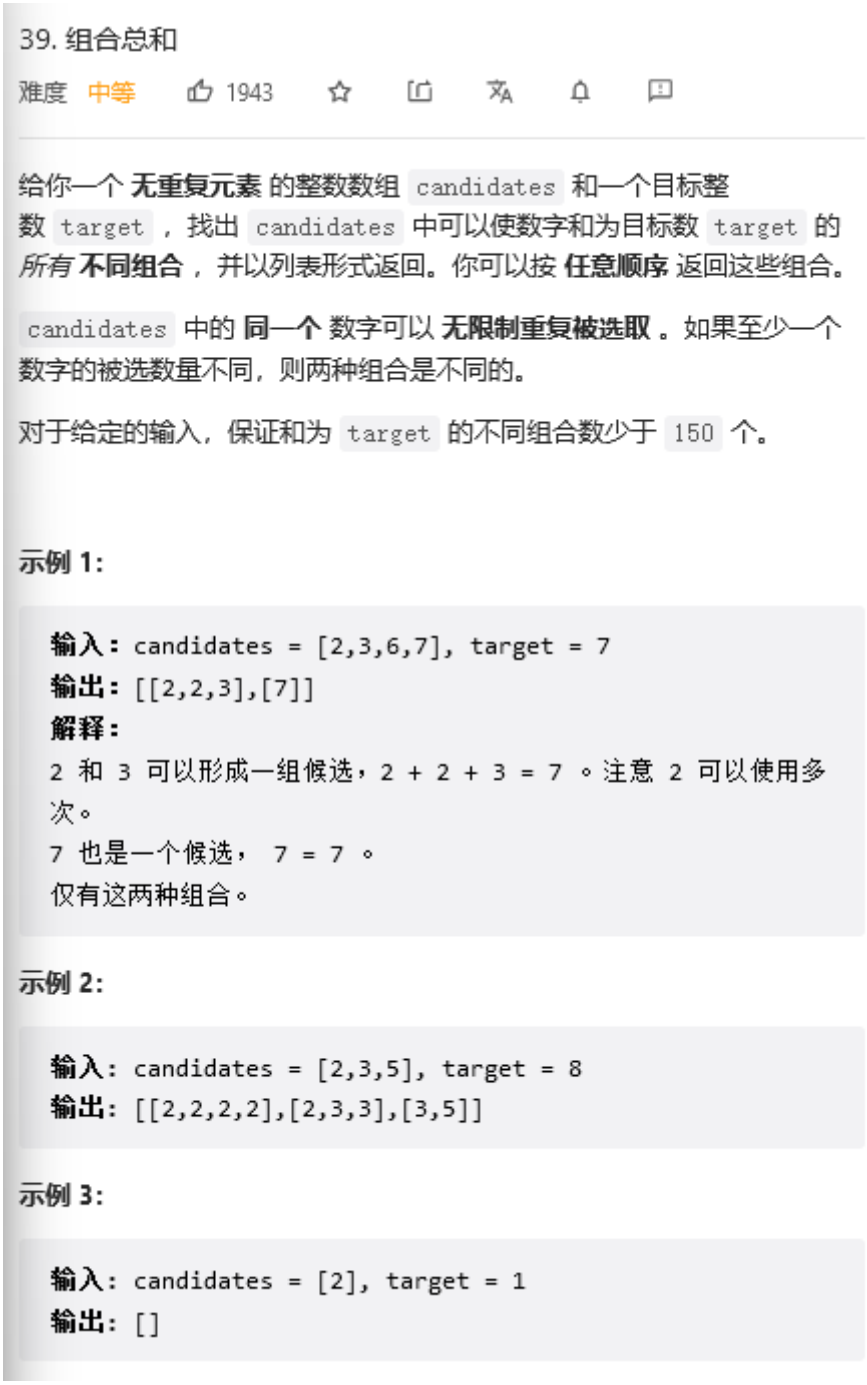
39-组合总和 I

题述

https://leetcode-cn.com/problems/combination-sum/submissions/



39. 组合总和

难度 中等 1943

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]]

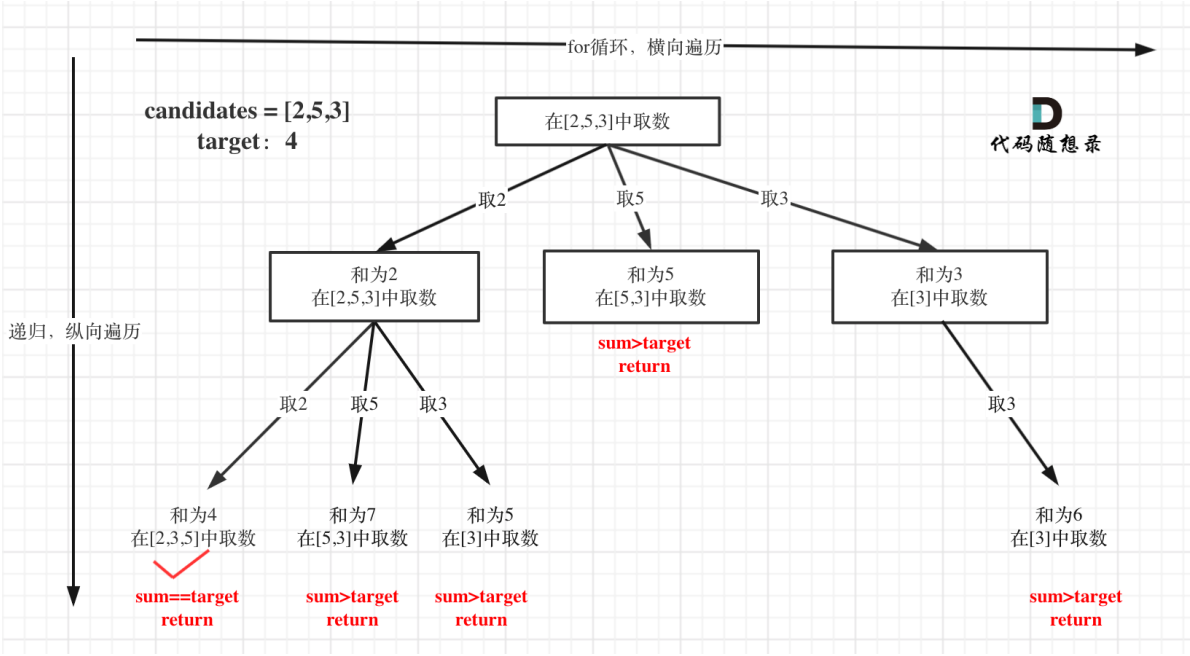
示例 3：

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

思路

注意到0的情况，万幸题目有要求，candidates中没有0

此外，注意到关键词：可无限制重复选取



本题唯一的限制便是加和为target的限制

回溯三板斧

- 递归函数参数
  - 这里依然是定义两个全局变量，二维数组result存放结果集，数组path存放符合条件的结果。（这两个变量可以作为函数参数传入）

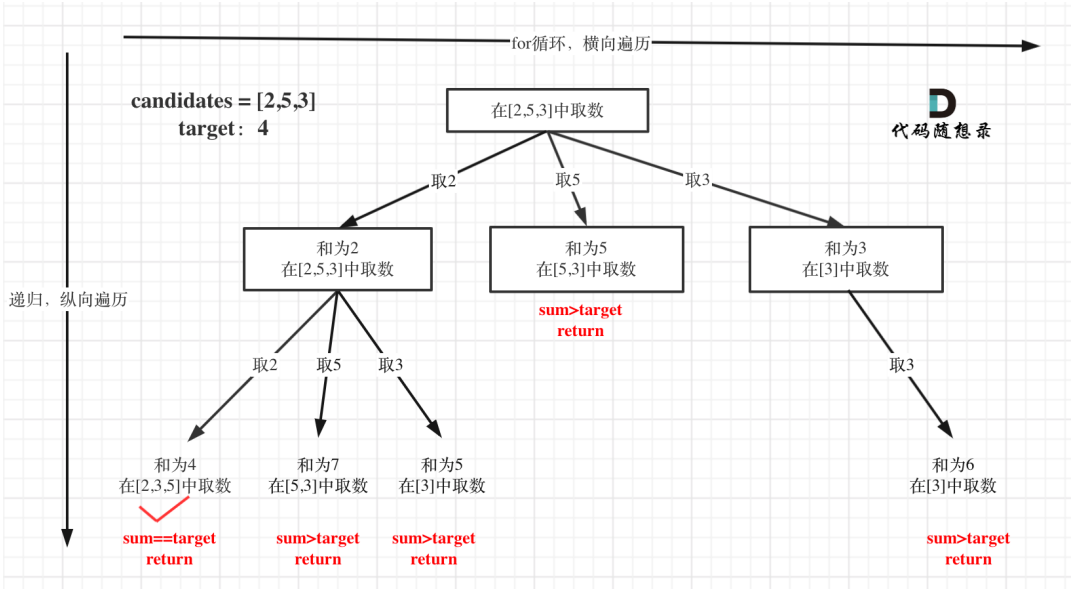
    首先是题目中给出的参数，集合candidates, 和目标值target。

    此外我还定义了int型的sum变量来统计单一结果path里的总和，其实这个sum也可以不用，用target做相应的减法就可以了，最后如何target==0就说明找到符合的结果了，但为了代码逻辑清晰，我依然用了sum。

    本题还需要startIndex来控制for循环的起始位置

```
vector<vector<int>> result;
vector<int> path;
void backtracking(vector<int>& candidates, int target, int sum, int
startIndex)
```

- 递归终止条件
  -



- 从叶子节点可以清晰看到，终止只有两种情况，sum大于target和sum等于target。
- sum等于target的时候，需要收集
- ```
  if (sum > target) {
      return;
  }
  if (sum == target) {
      result.push_back(path);
      return;
  }
  ```

- 单层搜索
  - 单层for循环依然是从startIndex开始，搜索candidates集合。
  - 本题元素可重复选取
  - startindex不用再i+1
  - ```
    for (int i = startIndex; i < candidates.size(); i++) {
        sum += candidates[i];
        path.push_back(candidates[i]);
        backtracking(candidates, target, sum, i); // 关键点:不用i+1了，表示可以重复读取当前的数
        sum -= candidates[i];   // 回溯
        path.pop_back();        // 回溯
    }
    ```

题解

C++-回溯

```
class Solution {
private:
    vector<vector<int>> result;
    vector<int> path;
    void backtracking(vector<int>& candidates, int target, int sum, int startIndex) {
        if (sum > target) {
            return;
        }
        if (sum == target) {
            result.push_back(path);
            return;
        }

        for (int i = startIndex; i < candidates.size(); i++) {
            sum += candidates[i];
            path.push_back(candidates[i]);
            backtracking(candidates, target, sum, i); // 不用i+1了，表示可以重复读取当前的数
            sum -= candidates[i];
            path.pop_back();
        }
    }
```

```
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        result.clear();
        path.clear();
        backtracking(candidates, target, 0, 0);
        return result;
    }
};
```

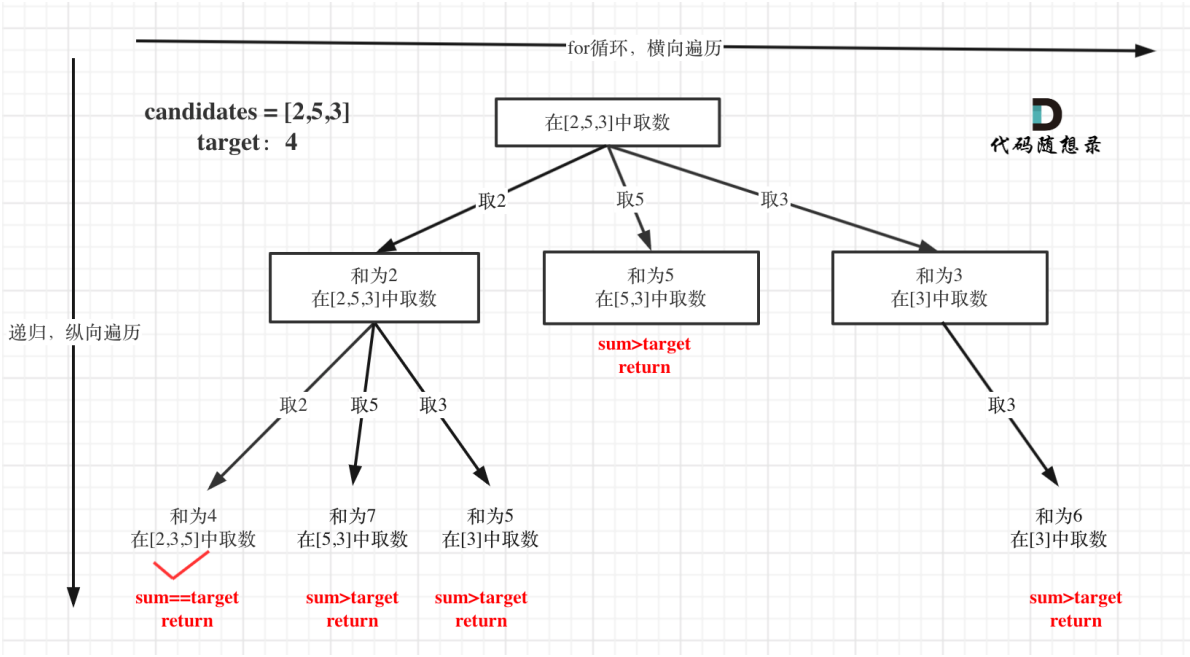
Python-回溯

```
class Solution:
    def __init__(self):
        self.path = []
        self.result = []

    def BackTracking(self,candidates: List[int], target: int, sum_: int,
start_index: int) -> None:
        if sum_ == target:
            self.result.append(self.path[:])    #深复制方式加入
            return
        if sum_ > target:
            return

        #单层递归
        for i in range(start_index,len(candidates)):
            sum_ += candidates[i]
            self.path.append(candidates[i])
            self.BackTracking(candidates,target,sum_,i)
            sum_ -= candidates[i]   #回溯
            self.path.pop() #回溯
    def combinationSum(self, candidates: List[int], target: int) ->
List[List[int]]:
        self.BackTracking(candidates,target,0,0)
        return self.result
```

剪枝优化

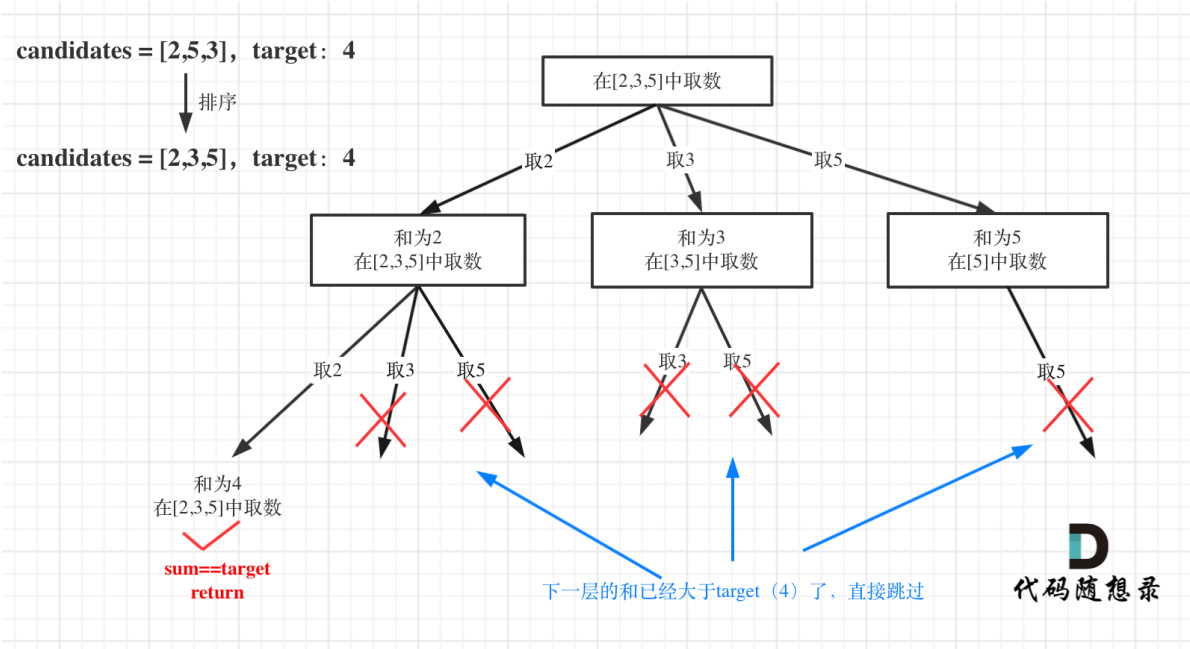


可以看到，对于sum已经大于target的情况，其实是依然进入了下一层递归，只是下一层递归结束判断的时候，会判断sum > target的话就返回。

其实如果已经知道下一层的sum会大于target，就没有必要进入下一层递归了。

那么可以在for循环的搜索范围上做做文章了。

对总集合排序之后，如果下一层的sum（就是本层的 sum + candidates[i]）已经大于target，就可以结束本轮for循环的遍历。



for循环剪枝代码如下：

```
for (int i = startIndex; i < candidates.size() && sum + candidates[i] <= target;
i++)
```

C++

```
class Solution {
private:
    vector<vector<int>> result;
    vector<int> path;
    void backtracking(vector<int>& candidates, int target, int sum, int
startIndex) {
        if (sum == target) {
            result.push_back(path);
            return;
        }

        // 如果 sum + candidates[i] > target 就终止遍历
        for (int i = startIndex; i < candidates.size() && sum + candidates[i] <=
target; i++) {
            sum += candidates[i];
            path.push_back(candidates[i]);
            backtracking(candidates, target, sum, i);
            sum -= candidates[i];
            path.pop_back();

        }
    }
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        result.clear();
        path.clear();
        sort(candidates.begin(), candidates.end()); // 需要排序
        backtracking(candidates, target, 0, 0);
        return result;
```

```
    }
};
```

思考

本题有两点不同：

- 组合没有数量要求
- 元素可无限重复选取