0039. 组合总和

ITCharge 大约 3 分钟

- 标签：数组、回溯
- 难度：中等

题目链接

- [0039. 组合总和 - 力扣]

题目大意

描述：给定一个无重复元素的正整数数组 `candidates` 和一个正整数 `target` 。

要求：找出 `candidates` 中所有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。可以按照任意顺序返回这些组合。

说明：

- 数组 `candidates` 中的数字可以无限重复选取。
- 如果至少一个数字的被选数量不同，则两种组合是不同的。
- $1 \le candidates.length \le 30$。
- $2 \le candidates[i] \le 40$。
- `candidates` 的所有元素互不相同。
- $1 \le target \le 40$。

示例：

- 示例 1：

```py
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：

```py
输入: candidates = [2,3,5], target = 8
输出: [[2,2,2,2],[2,3,3],[3,5]]
```

解题思路

思路 1：回溯算法

定义回溯方法，start_index = 1 开始进行回溯。

- 如果 `sum > target` ，则直接返回。
- 如果 `sum == target` ，则将 path 中的元素加入到 res 数组中。
- 然后对 `[start_index, n]` 范围内的数进行遍历取值。
  - 如果 `sum + candidates[i] > target` ，可以直接跳出循环。
  - 将和累积，即 `sum += candidates[i]` ，然后将当前元素 i 加入 path 数组。
  - 递归遍历 `[start_index, n]` 上的数。
  - 加之前的和回退，即 `sum -= candidates[i]` ，然后将遍历的 i 元素进行回退。
- 最终返回 res 数组。

根据回溯算法三步走，写出对应的回溯算法。

1. **明确所有选择**：一个组合每个位置上的元素都可以从剩余可选元素中选出。
2. **明确终止条件**：
   - 当遍历到决策树的叶子节点时，就终止了。即当前路径搜索到末尾时，递归终止。
3. **将决策树和终止条件翻译成代码：**
   1. 定义回溯函数：
      - `backtrack(total, start_index):` 函数的传入参数是 `total` （当前和）、 `start_index` （剩余可选元素开始位置），全局变量是 `res` （存放所有符合条件结果的集合数组）和 `path` （存放当前符合条件的结果）。
        - `backtrack(total, start_index):` 函数代表的含义是：当前组合和为 `total` ，递归从 `candidates` 的 `start_index` 位置开始，选择剩下的元素。
   2. 书写回溯函数主体（给出选择元素、递归搜索、撤销选择部分）。
      - 从当前正在考虑元素，到数组结束为止，枚举出所有可选的元素。对于每一个可选元素：
        - 约束条件：之前已经选择的元素不再重复选用，只能从剩余元素中选择。

- 选择元素：将其添加到当前数组 `path` 中。
- 递归搜索：在选择该元素的情况下，继续递归选择剩下元素。
- 撤销选择：将该元素从当前结果数组 `path` 中移除。

```py
for i in range(start_index, len(candidates)):
    if total + candidates[i] > target:
        break

    total += candidates[i]
    path.append(candidates[i])
    backtrack(total, i)
    total -= candidates[i]
    path.pop()
```

3. 明确递归终止条件（给出递归终止条件，以及递归终止时的处理方法）。
   - 当不可能再出现解（ `total > target` ），或者遍历到决策树的叶子节点时（ `total == target` ）时，就终止了。
   - 当遍历到决策树的叶子节点时（ `total == target` ）时，将当前结果的数组 `path` 放入答案数组 `res` 中，递归停止。

思路 1：代码

```py
class Solution:
    def combinationSum(self, candidates: List[int], target: int) ->
List[List[int]]:
        res = []
        path = []
        def backtrack(total, start_index):
            if total > target:
                return

            if total == target:
                res.append(path[:])
                return

            for i in range(start_index, len(candidates)):
                if total + candidates[i] > target:
                    break

                total += candidates[i]
                path.append(candidates[i])
```

```python
                backtrack(total, i)
                total -= candidates[i]
                path.pop()
        candidates.sort()
        backtrack(0, 0)
        return res
```

思路 1：复杂度分析

- **时间复杂度**：$O(2^n \times n)$，其中 n 是数组 `candidates` 的元素个数，2^n 指的是所有状态数。
- **空间复杂度**：$O(target)$，递归函数需要用到栈空间，栈空间取决于递归深度，最坏情况下递归深度为 $O(target)$，所以空间复杂度为 $O(target)$。