40. Combination Sum II

Created	@July 19, 2020 8:39 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/combination-sum-ii/
Importance	*****
Tag	Backtrack
Video	

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sum to `target`.

Each number in `candidates` may only be used **once** in the combination.

Note: The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]
```

Example 2:

```
Input: candidates = [2,5,2,1,2], target = 5
Output:
[
[1,2,2],
[5]
]
```

Constraints:

- `1 <= candidates.length <= 100`
- `1 <= candidates[i] <= 50`
- `1 <= target <= 30`

Solution

```
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        results = []
        subset = []

        if not candidates:
            return results

        candidates.sort()
        self.dfs(candidates, target, 0, subset, results)

        return results

    def dfs(self, candidates, target, index, subset, results):
        if target == 0:
            results.append(list(subset))
        elif target > 0:
            for i in range(index, len(candidates)):
                if i != index and candidates[i] == candidates[i - 1]:
                    continue
                subset.append(candidates[i])
                self.dfs(candidates, target - candidates[i], i + 1, subset, results)
                subset.pop()
```

```
class Solution {
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        List<List<Integer>> results = new ArrayList<List<Integer>>();
        List<Integer> curList = new ArrayList<Integer>();
        if (candidates == null) return results;
        Arrays.sort(candidates);
        helper(candidates, target, 0, curList, results);
        return results;
    }

    public void helper(int[] candidates, int target, int index, List<Integer> curList, List<List<Integer>> results) {
        if (target == 0) {
            results.add(new ArrayList<Integer>(curList));
        } else if (target > 0) {
            for (int i = index; i < candidates.length; i++) {
                if (i != index && candidates[i] == candidates[i-1]) continue;
                curList.add(candidates[i]);
                helper(candidates, target-candidates[i], i+1, curList, results);
                curList.remove(curList.size()-1);
            }
        }
    }
}
```

对比子集问题的解法，只要额外用一个 `trackSum` 变量记录回溯路径上的元素和，然后将 base case 改一改即可解决这道题：

```
List<List<Integer>> res = new LinkedList<>();
// 记录回溯的路径
LinkedList<Integer> track = new LinkedList<>();
// 记录 track 中的元素之和
int trackSum = 0;
```

```java
public List<List<Integer>> combinationSum2(int[] candidates, int target) {
    if (candidates.length == 0) {
        return res;
    }
    // 先排序，让相同的元素靠在一起
    Arrays.sort(candidates);
    backtrack(candidates, 0, target);
    return res;
}

// 回溯算法主函数
void backtrack(int[] nums, int start, int target) {
    // base case，达到目标和，找到符合条件的组合
    if (trackSum == target) {
        res.add(new LinkedList<>(track));
        return;
    }
    // base case，超过目标和，直接结束
    if (trackSum > target) {
        return;
    }

    // 回溯算法标准框架
    for (int i = start; i < nums.length; i++) {
        // 剪枝逻辑，值相同的树枝，只遍历第一条
        if (i > start && nums[i] == nums[i - 1]) {
            continue;
        }
        // 做选择
        track.add(nums[i]);
        trackSum += nums[i];
        // 递归遍历下一层回溯树
        backtrack(nums, i + 1, target);
        // 撤销选择
        track.removeLast();
        trackSum -= nums[i];
    }
}
```