宫水三叶的刷题日记

组合总和

Author ：宫水三叶
Date ：2021/10/07
QQ Group: 703311589
WeChat ：oaoaya

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

噔噔噔噔，这是公众号「宫水三叶的刷题日记」的原创专题「组合总和问题」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「组合总和问题」即可获取最新下载链接。

💡下面介绍使用本合集的最佳使用实践：

学习算法：

1. 打开在线目录（Github 版 & Gitee 版）；
2. 从侧边栏的类别目录找到「组合总和问题」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题‘
4. 拿到题号之后，回到本合集进行检索。

维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流 🍭🍭🍭

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **39. 组合总和** ，难度为 **中等**。

Tag：「回溯算法」、「DFS」、「组合总和问题」

给定一个无重复元素的数组 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 中的数字可以无限制重复被选取。

说明：

- 所有数字（包括 target）都是正整数。
- 解集不能包含重复的组合。

示例 1：

```
输入：candidates = [2,3,6,7], target = 7,

所求解集为：
[
  [7],
  [2,2,3]
]
```

示例 2：

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

所求解集为：
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]
```

提示：

- 1 <= candidates.length <= 30
- 1 <= candidates[i] <= 200
- candidate 中的每个元素都是独一无二的。
- 1 <= target <= 500

DFS + 回溯

这道题很明显就是在考察回溯算法。

还记得三叶之前跟你分享过的 37. 解数独（困难） 吗？

里面有提到我们应该如何快速判断一道题是否应该使用 DFS + 回溯算法来爆搜。

总的来说，你可以从两个方面来考虑：

- **1. 求的是所有的方案，而不是方案数。** 由于求的是所有方案，不可能有什么特别的

优化，我们只能进行枚举。这时候可能的解法有动态规划、记忆化搜索、DFS + 回溯算法。

- **2. 通常数据范围不会太大，只有几十。** 如果是动态规划或是记忆化搜索的题的话，由于它们的特点在于低重复/不重复枚举，所以一般数据范围可以出到 10^5 到 10^7，而 DFS + 回溯的话，通常会限制在 30 以内。

这道题数据范围是 30 以内，而且是求所有方案，因此我们使用 DFS + 回溯来求解。

代码：

```
class Solution {
    public List<List<Integer>> combinationSum(int[] cs, int t) {
        List<List<Integer>> ans = new ArrayList<>();
        List<Integer> cur = new ArrayList<>();
        dfs(cs, t, 0, ans, cur);
        return ans;
    }

    /**
     * cs: 原数组，从该数组进行选数
     * t: 还剩多少值需要凑成。起始值为 target ，代表还没选择任何数；当 t = 0，代表选择的数凑成了 targe
     * u: 当前决策到 cs[] 中的第几位
     * ans: 最终结果集
     * cur: 当前结果集
     */
    void dfs(int[] cs, int t, int u, List<List<Integer>> ans, List<Integer> cur) {
        if (t == 0) {
            ans.add(new ArrayList<>(cur));
            return;
        }
        if (u == cs.length || t < 0) return;

        // 枚举 cs[u] 的使用次数
        for (int i = 0; cs[u] * i <= t; i++) {
            dfs(cs, t - cs[u] * i, u + 1, ans, cur);
            cur.add(cs[u]);
        }
        // 进行回溯。注意回溯总是将数组的最后一位弹出
        for (int i = 0; cs[u] * i <= t; i++) {
            cur.remove(cur.size() - 1);
        }
    }
}
```

- 时间复杂度：由于每个数字的使用次数不确定，因此无法分析具体的复杂度。但是 DFS 回溯算法通常是指数级别的复杂度（因此数据范围通常为 30 以内）。这里暂定 $O(n * 2^n)$
- 空间复杂度：同上。复杂度为 $O(n * 2^n)$

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **40. 组合总和 II** ，难度为 **中等**。

Tag：「回溯算法」、「DFS」、「组合总和问题」

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。

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]
]
```

DFS + 回溯

这道题和「39. 组合总和（中等）」几乎一样。

唯一的不同是这题每个数只能使用一次，而「39. 组合总和（中等）」中可以使用无限次。

我们再来回顾一下应该如何快速判断一道题是否应该使用 DFS + 回溯算法来爆搜。

这个判断方法，最早三叶在 37. 解数独（困难） 讲过。

总的来说，你可以从两个方面来考虑：

- **1. 求的是所有的方案，而不是方案数。** 由于求的是所有方案，不可能有什么特别的优化，我们只能进行枚举。这时候可能的解法有动态规划、记忆化搜索、DFS + 回溯算法。
- **2. 通常数据范围不会太大，只有几十。** 如果是动态规划或是记忆化搜索的题的话，由于它们的特点在于低重复/不重复枚举，所以一般数据范围可以出到 10^5 到 10^7，而 DFS + 回溯的话，通常会限制在 30 以内。

这道题数据范围是 30 以内，而且是求所有方案。因此我们使用 DFS + 回溯来求解。

我们可以接着 39. 组合总和（中等） 的思路来修改：

1. 由于每个数字只能使用一次，我们可以直接在 DFS 中决策某个数是用还是不用。
2. 由于不允许重复答案，可以使用 set 来保存所有合法方案，最终再转为 list 进行返回。当然我们需要先对 cs 进行排序，确保得到的合法方案中数值都是从小到大的。这样 set 才能起到去重的作用。对于 `[1,2,1]` 和 `[1,1,2]`，set 不会认为是相同的数组。

代码：

```java
class Solution {
    public List<List<Integer>> combinationSum2(int[] cs, int t) {
        Arrays.sort(cs);
        Set<List<Integer>> ans = new HashSet<>();
        List<Integer> cur = new ArrayList<>();
        dfs(cs, t, 0, ans, cur);
        return new ArrayList<>(ans);
    }

    /**
     * cs: 原数组，从该数组进行选数
     * t: 还剩多少值需要凑成。起始值为 target ，代表还没选择任何数；当 t = 0，代表选择的数凑成了 targe
     * u: 当前决策到 cs[] 中的第几位
     * ans: 最终结果集
     * cur: 当前结果集
     */
    void dfs(int[] cs, int t, int u, Set<List<Integer>> ans, List<Integer> cur) {
        if (t == 0) {
            ans.add(new ArrayList<>(cur));
            return;
        }
        if (u == cs.length || t < 0) return;

        // 使用 cs[u]
        cur.add(cs[u]);
        dfs(cs, t - cs[u], u + 1, ans, cur);

        // 进行回溯
        cur.remove(cur.size() - 1);
        // 不使用 cs[u]
        dfs(cs, t, u + 1, ans, cur);
    }
}
```

- 时间复杂度：DFS 回溯算法通常是指数级别的复杂度（因此数据范围通常为 30 以内）。这里暂定 $O(n * 2^n)$
- 空间复杂度：同上。复杂度为 $O(n * 2^n)$

💡更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。

最新专题合集资料下载，可关注公众号「宫水三叶的刷题日记」，回台回复「组合总和问题」获取下载链接。

觉得专题不错，可以请作者吃糖 🍭🍭🍭 ：