回溯算法 02

♪² 39. 组合总和

中等 15min

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: []
```

提示：

- `1 <= candidates.length <= 30`
- `2 <= candidates[i] <= 40`
- `candidates` 的所有元素 **互不相同**
- `1 <= target <= 40`

♪³ 题解

> 这道题感觉和之前的组合也是非常之相似，也就是说只要把startindex都设为0就行

上面的思想是非常错误的，因为做了之后发现这样就变成排列问题了，不能做排列问题啊，仔细想想发现是startindex不要加一，参上

回顾一下之前的三个步骤

- 参数和函数
- 返回条件
- 层内遍历逻辑

```cpp
class Solution {
public:
    vector<vector<int>> ans;
    vector<int> path;
    void find(vector<int>& candidates,int target,int start,int sum){//和组合问题一样，为了避免重复需要加start
        if(sum>=target){
            if(sum==target)ans.push_back(path);
            return;
        }
        for(int i=start;i<candidates.size();i++){
            path.push_back(candidates[i]);
            find(candidates,target,i,sum+candidates[i]);//这里的i不加1，表示可以重复选取当前元素
            path.pop_back();
        }
        return;
    }
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        find(candidates,target,0,0);
        return ans;
    }
};
```

♪² 40.组合总和II

中等 30min

力扣题目链接

给定一个候选人编号的集合 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** 。

注意：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,
输出:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]
```

示例 2:

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

提示:

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

♪³ 题解

说白了没太整明白，稀里糊涂做完的

关键就在于去重，这个怎么做到的捏

- 给数组排序
- 排完序之后，用used数组来标识再当前路径上有没有用过这个元素
- 之后用`if (i > 0 && candidates[i] == candidates[i - 1] && used[i - 1] == false)`来判断，第一个判断是否相同，第二个判断这个相同的元素在当前路径中有没有用过，如果用过就说明即使再选了只是这个元素会重复，而出现的情况不会相同，如果没用过就说明再选这个就会再选和之前相同的情况。

```
class Solution {
private:
    vector<vector<int>> result;
    vector<int> path;
    void backtracking(vector<int>& candidates, int target, int sum, int startIndex,
vector<bool>& used) {
        if (sum == target) {
            result.push_back(path);
            return;
        }
        for (int i = startIndex; i < candidates.size() && sum + candidates[i] <= target; i++)
{
            // used[i - 1] == true,说明同一树枝candidates[i - 1]使用过
            // used[i - 1] == false,说明同一树层candidates[i - 1]使用过
```

```
            // 要对同一树层使用过的元素进行跳过
            if (i > 0 && candidates[i] == candidates[i - 1] && used[i - 1] == false) {
                continue;
            }
            sum += candidates[i];
            path.push_back(candidates[i]);
            used[i] = true;
            backtracking(candidates, target, sum, i + 1, used); // 和39.组合总和的区别1,这里是i+1,每个数字在每个组合中只能使用一次
            used[i] = false;
            sum -= candidates[i];
            path.pop_back();
        }
    }

public:
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        vector<bool> used(candidates.size(), false);
        path.clear();
        result.clear();
        // 首先把给candidates排序,让其相同的元素都挨在一起。
        sort(candidates.begin(), candidates.end());
        backtracking(candidates, target, 0, 0, used);
        return result;
    }
};
```

♪² 131.分割回文串

中等

力扣题目链接

给你一个字符串 `s`,请你将 `s` 分割成一些 子串,使每个子串都是 **回文串** 。返回 `s` 所有可能的分割方案。

示例 1:

```
输入:s = "aab"
输出:[["a","a","b"],["aa","b"]]
```

示例 2:

```
输入:s = "a"
输出:[["a"]]
```

提示:

- `1 <= s.length <= 16`
- `s` 仅由小写英文字母组成

♪³ 题解

我浅薄的想法认为可以逐步划分，也就是从左到右依次加一个字符，如果是回文串则递归处理剩下的，不是的话就直接返回，终止条件就是字符串被处理完。

```
class Solution {
public:
    vector<string> sub;
    vector<vector<string>> ans;
    bool isHuiwen(string s){//判断是不是回文
        int n=s.size();
        for(int i=0;i<=n/2;i++){//看看相等不
            if(s[i]!=s[n-i-1])return false;
        }
        return true;
    }
    void find(string s,int start){//回溯
        if(start==s.size()){
            ans.push_back(sub);
            return;
        }
        for(int i=start;i<s.size();i++)
        {
            string p=s.substr(start,i-start+1);//一个小函数，第一个参数是开始位置，第二个参数是长度，这边也是得复习一下了
            if(!isHuiwen(p))continue;//如果不是回文就不能划分，直接跳过
            sub.push_back(p);//把这个划分加入
            find(s,i+1);//继续处理剩下的子串
            sub.pop_back();//回溯
        }
    }
    vector<vector<string>> partition(string s) {
        find(s,0);
        return ans;
    }
};
```

♪² 附录：string 成员函数汇总表

下面是一个常见的 std::string 成员函数的汇总：

函数名	描述	示例代码
size()	返回字符串的长度（字符数）。	std::cout << str.size();
length()	与 size() 相同，返回字符串的长度。	std::cout << str.length();
empty()	判断字符串是否为空。	std::cout << (str.empty() ? "Yes" : "No");
operator[]	访问字符串中指定位置的字符。	std::cout << str[0];

函数名	描述	示例代码
at()	访问字符串中指定位置的字符(带边界检查)。	`std::cout << str.at(0);`
substr()	返回从指定位置开始的子字符串。	`std::string sub = str.substr(0, 5);`
find()	查找子字符串在字符串中的位置。	`std::cout << str.find("sub") << std::endl;`
rfind()	从字符串末尾开始查找子字符串的位置。	`std::cout << str.rfind("sub") << std::endl;`
replace()	替换字符串中的部分内容。	`str.replace(pos, length, "new_substring");`
append()	在字符串末尾添加内容。	`str.append(" more");`
insert()	在指定位置插入内容。	`str.insert(pos, "inserted");`
erase()	删除指定位置的字符或子字符串。	`str.erase(pos, length);`
clear()	清空字符串。	`str.clear();`
c_str()	返回 C 风格的字符串(以 null 结尾)。	`const char* cstr = str.c_str();`
data()	返回指向字符数据的指针(C++11 及之后的版本)。	`const char* data = str.data();`
compare()	比较两个字符串。	`int result = str.compare("other");`
find_first_of()	查找第一个匹配任意字符的位置。	`size_t pos = str.find_first_of("aeiou");`
find_last_of()	查找最后一个匹配任意字符的位置。	`size_t pos = str.find_last_of("aeiou");`
find_first_not_of()	查找第一个不匹配任意字符的位置。	`size_t pos = str.find_first_not_of("aeiou");`
find_last_not_of()	查找最后一个不匹配任意字符的位置。	`size_t pos = str.find_last_not_of("aeiou");`