39、 Combination Sum

Given a **set** of candidate numbers (**C**) **(without duplicates)** and a target number (**T**), find all unique combinations in **C** where the candidate numbers sums to **T**.

The **same** repeated number may be chosen from **C** unlimited number of times.

Note:

- All numbers (including target) will be positive integers.
- The solution set must not contain duplicate combinations.

For example, given candidate set `[2, 3, 6, 7]` and target `7`, A solution set is:

```
[
  [7],
  [2, 2, 3]
]
```

【思路】

DFS，递归。

因为不能有重复答案，所以我们使用递归时需要加一些约束条件来去重。首先，答案中每一个vector元素是有序的，所以我们先考虑排序。另外，排序也可以起到方便去重作用。

我们可以设需要解决的问题为：P[target]，对于candidate内的元素a，假如a<target， 那么解的一个组合中就可以包含一个a，问题进一步成为解决P[target-a]，我么递归地将target减去candidate中元素，直到为0就可以得到解的一个组合。

那么我们应该怎么去重呢？

第一，序列已经排好序了。

第二，顺序标记序列中元素，已经标记的元素不能作为访问元素来达到去重的目的。

```cpp
class Solution {
public:
    vector <vector<int>> res;
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        sort(candidates.begin(), candidates.end());
        DFS(candidates, target, vector<int>());
        return res;
    }
private:
    void DFS(vector<int>& candidates, int target, vector<int> remain){
        if(target == 0){
            res.push_back(remain);
            return;
        }
        for(int i = 0; i < candidates.size(); i++){
            if(candidates[i] > target){
                break;
            }
            if(remain.empty() || candidates[i] >= remain[remain.size()-1]){
                remain.push_back(candidates[i]);
                DFS(candidates, target - candidates[i], remain);
                remain.pop_back();
            }
        }
    }
};
```