回溯算法

回溯法就是暴力搜索

第39题. 组合总和

题目链接：https://leetcode-cn.com/problems/combination-sum/

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。

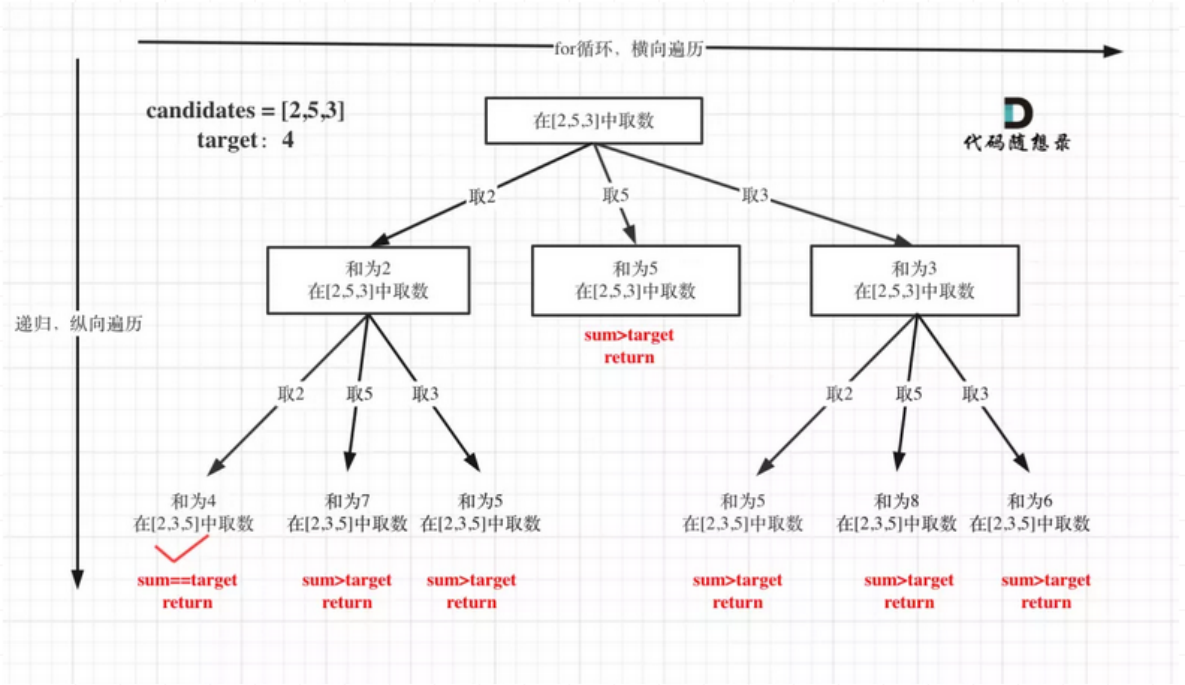
candidates 中的数字可以无限制重复被选取。

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。

本题目中的核心在于可以**无限重复地选取**

本题搜索的过程抽象成树形结构如下：



本题还需要startIndex来控制for循环的起始位置，**对于组合问题，什么时候需要startIndex呢？如果是一个集合来求组合的话，就需要用startIndex，如果是多个集合求组合，各个集合之间相互不影响.**

```
vetor<vector<int>> result;
```

```
vector<int> path;
void backtracking(vector<int>& condidates,int target,int sum,int startIndex);
```

从叶子节点可以清晰看到，终止只有两种情况，sum大于target和sum等于target

```
if(sum>target){
 return ;
}
if(sum==target){
 result.push_back(path);
 return ;
}
```

如何满足可重复选取的条件呢?

```
for(int i=startIndex;i<candidates.size();i++){
 path.push_back(candidates[i]);
 sum+=candidates[i];
 backtracking(candidates,target,sum,i);
 sum-=candidates[i];
 path.pop_back();
}
```

进行剪枝操作?

如果下一层的sum已经大于target,就可以结束本轮for循环的遍历方式

```
for(int i=startIndex;i<candidates.size()&&sum+candidates[i]<=target;i++){
 path.push_back(candidates[i]);
 sum+=candidates[i];
 backtracking(candidates,target,sum,i);
 sum-=candidates[i];
 path.pop_back();
}
```

本题和我们之前经过的不同之处在于

组合数量没有要求，元素可无限重复选取，所以在单层逻辑中遍历从startIndex开始

```
class solution{
 private:
 vector<vector<int>> result;
 vector<int> path;
 void backtracking(vector<int>& candidates,int count,int startIndex){
 if(count==0){
 result.push_back(path);
 return ;
 }
  //确定单层逻辑
```

```
    for(int i=startIndex,i<candidates.size()&&(count-candidates[i])>=0;i++)
{
    path.push_back(candidates[i]);
    backtracking(candidates,count-candidates[i],i);
    path.pop_back();
    }
    }
    public:
    vector<vector<int>> combinationSum(vector<int>& candidates,int target){
    if(candidates.empty()) return result;
    //此时需要排序
    sort(candidates.begin(),candidates.end());
    backtracking(candidates,target,0);
    return result;
    }
};
```

在求和问题中，**排序之和加剪枝**是**常见的套路**

40.组合总和II

题目链接：https://leetcode-cn.com/problems/combination-sum-ii/

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]
]

candidates中的每个数字在每个组合中只能使用一次，则说明需要无重复，
则**对所求的数字先进行排序**
该题目与前面一道题的区别在于：

1、本题的candidates中的每个数字在每个组合中只能使用一次
2、本题数组candidates的元素是可重复的，而39题中元素不能重复

如何理解去重?

所谓去重，就是**使用过的元素不能重复使用.**

组合问题可以理解为树形结构问题，那么使用过就有两个维度上
1、同一树枝上使用过
2、同一树层上使用过

本题目中一个组合中的元素可以重复，但是组合不能相同，则去重的是同一树层不能重复。

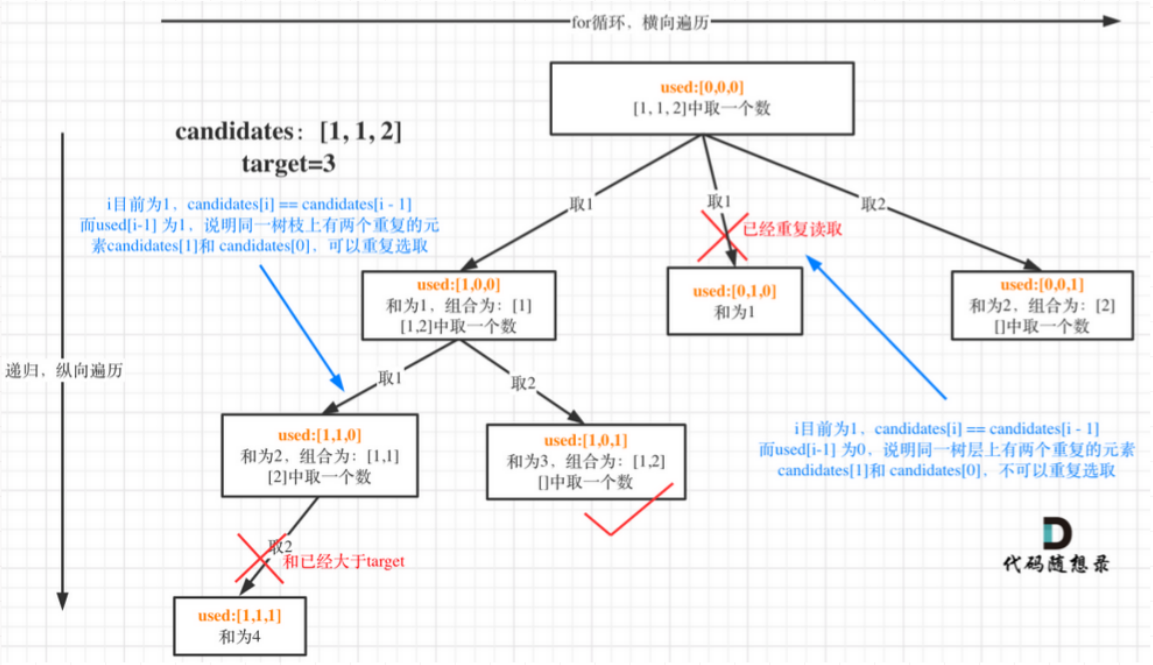
step1：递归函数参数
此时还需要一个bool型数组参数used，用来**记录同一个树枝上是否使用过**

```
vector<int> path;
vector<vector<int>> result;
void backtracking(vector<int>& candidates,int count,int startIndex)
step2:递归终止条件
if(count==0){
  result.push_back(path);
  return ;
}
step3:单层搜索逻辑
当candidates[i]==candidates[i-1]时
   used[i-1]=false;//则说明同一个树层使用过
   used[i-1]=true;//则说明同一个树枝上使用过
for(int i=startIndex;i<candidates.size()&&count-candidates[i]>=0;i++){

     if(i>0&&candidates[i]==candidates[i-1]&&used[i-1]==false)continue;
     path.push_back(candidates[i]);
     used[i]=true;
     backtracking(candidates,count-candidates[i],i+1);
     used[i]=false;
     path.pop_back();
}
```

此时for循环里就应该做continue的操作。

这块比较抽象，如图：



我在图中将used的变化用橘黄色标注上，可以看出在candidates[i] == candidates[i - 1]相同的情况下：

然后回溯入口为
sort(candidates.begin(),candidates.end())
backtracking(candidates,target,0)

所以Carl有必要把去重的这块彻彻底底的给大家讲清楚，**「就连“树层去重“和“树枝去重”都是我自创的词汇，希望对大家理解有帮助！」**