回溯

解题思路：

1.DFS 和回溯算法区别

DFS 是一个劲的往某一个方向搜索，而回溯算法建立在 DFS 基础之上的，但不同的是在搜索过程中，达到结束条件后，恢复状态，回溯上一层，再次搜索。因此回溯算法与 DFS 的区别就是有无状态重置。

2.何时使用回溯算法

当问题需要 "回头"，以此来查找出所有的解的时候，使用回溯算法。即满足结束条件或者发现不是正确路径的时候(走不通)，要撤销选择，回退到上一个状态，继续尝试，直到找出所有解为止。

3.怎么样写回溯算法(从上而下，※代表难点，根据题目而变化)

①画出递归树，找到状态变量(回溯函数的参数)，这一步非常重要※

②根据题意，确立结束条件

③找准选择列表(与函数参数相关),与第一步紧密关联※

④判断是否需要剪枝

⑤作出选择，递归调用，进入下一层

⑥撤销选择

4.回溯问题的类型

这里先给出，我总结的回溯问题类型，并给出相应的 leetcode 题目(一直更新)，然后再说如何去编写。**特别关注搜索类型的**，搜索类的搞懂，你就真的搞懂回溯算法了,，是前面两类是基础，帮助你培养思维

类型	题目

子集、组合	子集、子集 II、组合、组合总和、组合总和 II
全排列	全排列、全排列 II、字符串的全排列、字母大小写全排列
搜索	解数独、单词搜索、N 皇后、分割回文串、二进制手表

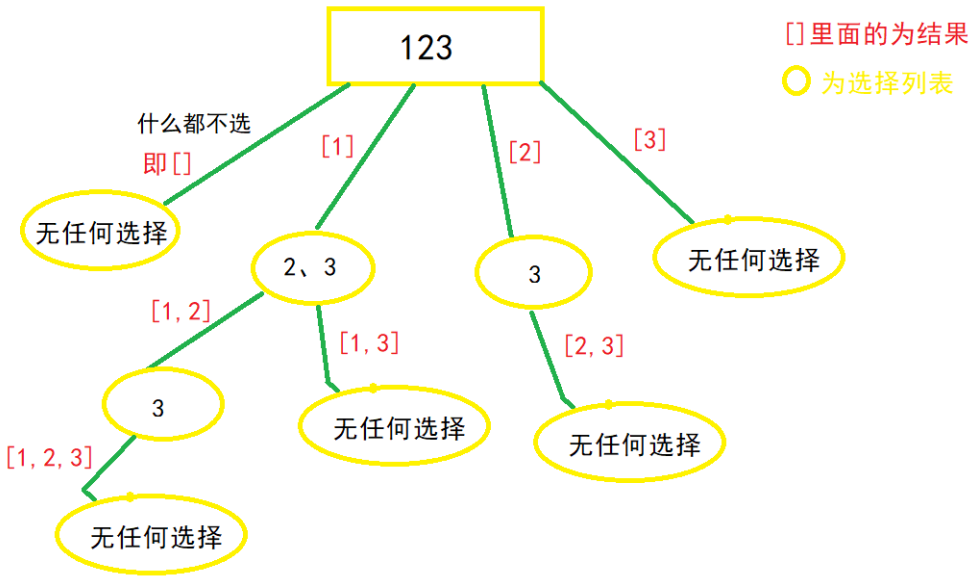
注意：子集、组合与排列是不同性质的概念。子集、组合是无关顺序的，而排列是和元素顺序有关的，如 [1，2] 和 [2，1] 是同一个组合(子集)，但 [1,2] 和 [2,1] 是两种不一样的排列！！！！因此被分为两类问题

5.回到子集、组合类型问题上来(ABC 三道例题)

A、 子集 - 给定一组不含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。

解题步骤如下：

①递归树



观察上图可得，选择列表里的数，都是选择路径(红色框)后面的数，比如[1]这条路径，

他后面的选择列表只有"2、3"，[2]这条路径后面只有"3"这个选择，那么这个时候，就应该使用一个参数 start，来标识当前的选择列表的起始位置。也就是标识每一层的状态，因此被形象的称为"状态变量",最终函数签名如下：

```
//nums 为题目中的给的数组
//path 为路径结果，要把每一条 path 加入结果集
void backtrack(vector<int>nums,vector<int>&path,int start)
```

②找结束条件

此题非常特殊，所有路径都应该加入结果集，所以不存在结束条件。或者说当 start 参数越过数组边界的时候，程序就自己跳过下一层递归了，因此不需要手写结束条件,直接加入结果集

```
**res 为结果集，是全局变量 vector<vector<int>>res,到时候要返回的
res.push_back(path);//把每一条路径加入结果集
```

③找选择列表

在①中已经提到过了，子集问题的选择列表，是上一条选择路径之后的数,即

```
for(int i=start;i<nums.size();i++)
```

④判断是否需要剪枝

从递归树中看到，路径没有重复的，也没有不符合条件的，所以不需要剪枝

⑤做出选择(即 for 循环里面的)

```
void backtrack(vector<int>nums,vector<int>&path,int start)
{
    for(int i=start;i<nums.size();i++)
    {
        //做出选择
        path.push_back(nums[i]);
        //递归进入下一层，注意 i+1，标识下一个选择列表的开始位置，最重要的一步
```

```
            backtrack(nums,path,i+1);
        }
}
```

⑥撤销选择

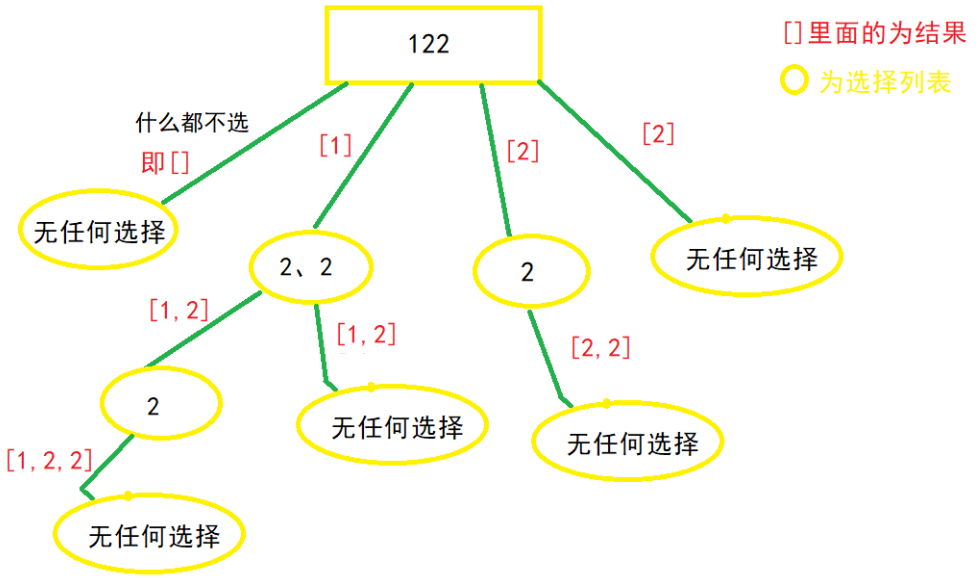
整体的 backtrack 函数如下：

```
    void backtrack(vector<int>nums,vector<int>&path,int start)
    {
        res.push_back(path);
        for(int i=start;i<nums.size();i++)
        {
            path.push_back(nums[i]);
            backtrack(nums,path,i+1);
            //撤销选择
            path.pop_back();
        }
    }
```

B、**子集 II**(剪枝思想)--问题描述：

给定一个可能 **包含重复元素** 的整数数组 nums，**返回**该数组所有可能的子集（幂集）。

①递归树

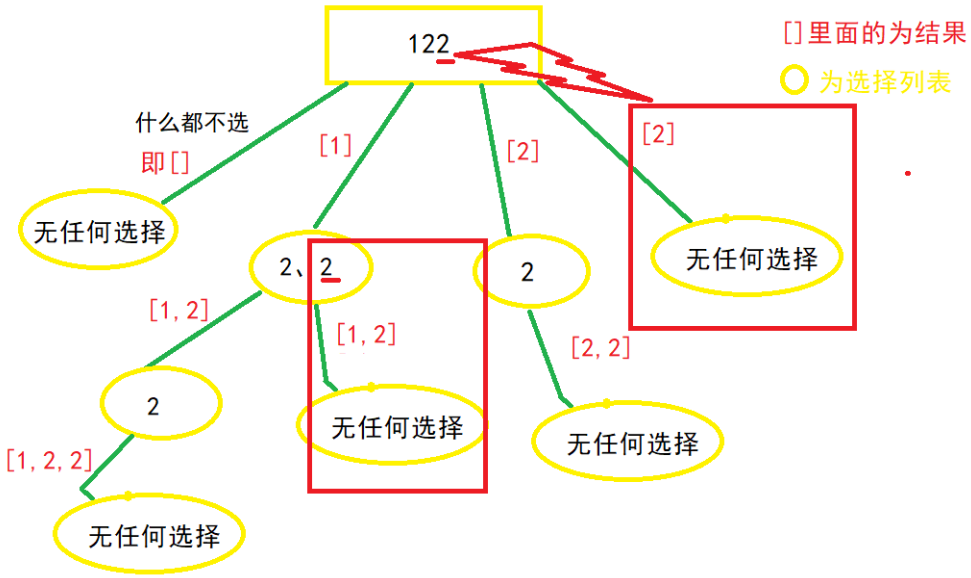


可以发现，树中出现了大量重复的集合，②**和**③**和第一个**问题一样，不再赘述，我们直接看第四步

④判断是否需要剪枝，需要先对数组排序，使用排序函数 `sort(nums.begin(),nums.end())`

显然我们需要去除重复的集合，即需要剪枝，把递归树上的某些分支剪掉。**那么**应去除哪些分支呢？又该如何编码呢？

观察上图不难发现，应该去除当前选择列表中，与上一个数重复的那个数，引出的分支，如 “2，2” 这个选择列表，第二个 “2” 是最后重复的，应该去除这个 “2” 引出的分支



(**去除**图中红色大框中的分支)

编码呢，刚刚说到是“去除当前选择列表中，与上一个数重复的那个数，引出的分支”，说明当前列表最少有两个数，当 i>start 时，做选择的之前，比较一下当前数，与上一个数 (i-1) 是不是相同，相同则 continue

```
void backtrack(vector<int>& nums,vector<int>&path,int start)
    {
        res.push_back(path);
        for(int i=start;i<nums.size();i++)
        {
            if(i>start&&nums[i]==nums[i-1])     //剪枝去重
                continue;
        }
    }
```

⑤做出选择

```
void backtrack(vector<int>& nums,vector<int>&path,int start)
    {
```

```
        res.push_back(path);
        for(int i=start;i<nums.size();i++)
        {
            if(i>start&&nums[i]==nums[i-1])      //剪枝去重
                continue;
            temp.push_back(nums[i]);
            backtrack(nums,path,i+1);
        }
    }
```

⑥撤销选择

整体的 backtrack 函数如下：

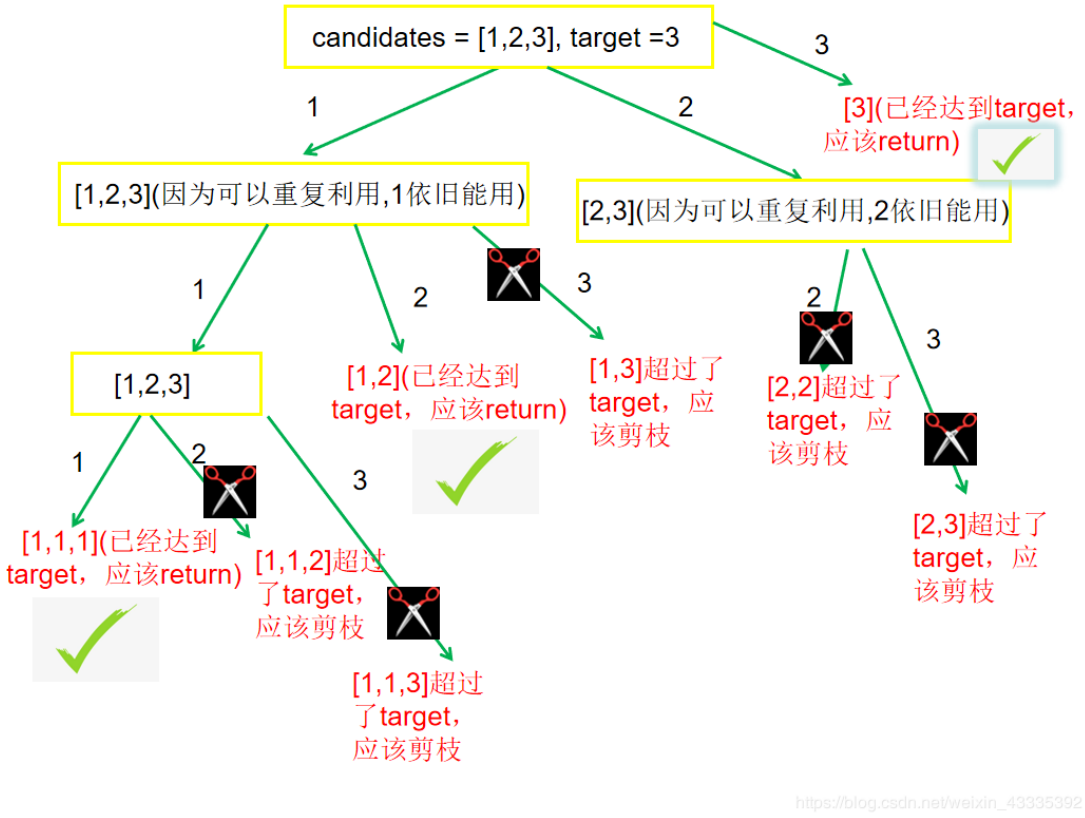
```
** sort(nums.begin(),nums.end());
void backtrack(vector<int>& nums,vector<int>&path,int start)
    {
        res.push_back(path);
        for(int i=start;i<nums.size();i++)
        {
            if(i>start&&nums[i]==nums[i-1])//剪枝去重
                continue;
            temp.push_back(nums[i]);
            backtrack(nums,path,i+1);
            temp.pop_back();
        }
    }
```

C、组合总和 - 问题描述

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。**

①递归树



(绿色箭头上面的是路径，红色框[]则为结果，黄色框为选择列表)

从上图看出，组合问题和子集问题一样，1,2 和 2,1 是同一个组合，因此 需要引入 start 参数标识，每个状态中选择列表的起始位置。另外，每个状态还需要一个 sum 变量，来记录当前路径的和，函数签名如下：

```
void backtrack(vector<int>& nums,vector<int>&path,int start,int sum,int target)
```

②找结束条件

由题意可得，当路径总和等于 target 时候，就应该把路径加入结果集，并 return

```
    if(target==sum){
```

```
            res.push_back(path);
            return;
        }
```

③找选择列表

```
for(int i=start;i<nums.size();i++)
```

④判断是否需要剪枝

从①中的递归树中发现，当前状态的 sum 大于 target 的时候就应该剪枝，不用再递归下去了

```
            for(int i=start;i<nums.size();i++){
            //剪枝
                    if(sum>target)continue;
            }
```

⑤做出选择

题中说数可以无限次被选择，那么 *i* 就不用 *+1* 。即下一层的选择列表，从自身开始。并且要更新当前状态的 sum

```
            for(int i=start;i<nums.size();i++){
                    if(sum>target)
                        continue;
                    path.push_back(nums[i]);
                    backtrack(nums,path,i,sum+nums[i],target);
            //i 不用+1(重复利用)，并更新当前状态的 sum
            }
```

⑤撤销选择

整体的 backtrack 函数如下：

```
void backtrack(vector<int>& nums,vector<int>&path,int start,int sum,int target)
    {
        for(int i=start;i<nums.size();i++)
                {
                    if(sum>target)    continue;
                    path.push_back(nums[i]);
                    backtrack(nums,path,i,sum+nums[i],target);
                    pacht.pop_back();
                }
    }
```

总结：子集、组合类问题，关键是用一个 start 参数来控制选择列表！！最后回溯六步走：

①画出递归树，找到状态变量(回溯函数的参数)，这一步非常重要※

②根据题意，确立结束条件

③找准选择列表(与函数参数相关),与第一步紧密关联※

④判断是否需要剪枝

⑤作出选择，递归调用，进入下一层

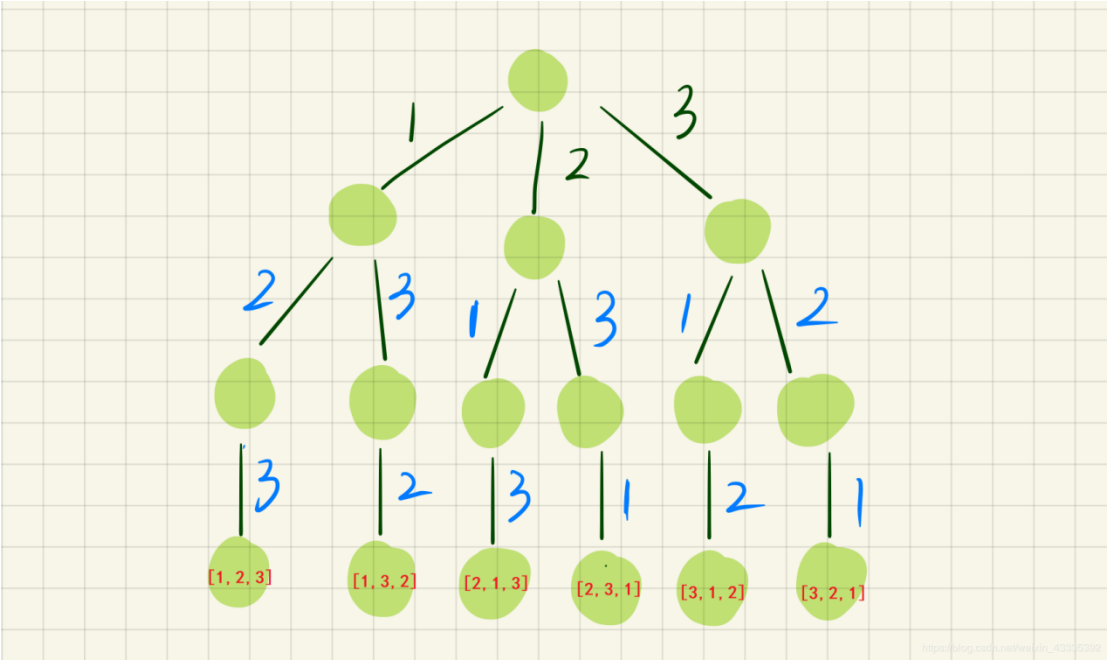
⑥撤销选择

排列类型(ABC 三道例题)

A. 全排列--问题描述

给定一个 **没有重复** 数字的序列，**返回其所有可能的全排列。**

①递归树



然后我们来回想一下，整个问题的思考过程，这棵树是如何画出来的。首先，我们固定 1，然后只有 2、3 可选：如果选 2，那就只剩 3 可选，得出结果[1,2,3]；如果选 3，那就只剩 2 可选，得出结果[1,3,2]。再来，如果固定 2，那么只有 1,3 可选：如果选 1，那就只剩 3，得出结果[2,1,3].....

有没有发现一个规律：如果我们固定了(选择了)某个数，那么他的下一层的选择列表就是——除去这个数以外的其他数！！比如，第一次选择了 2，那么他的下一层的选择列表只有 1 和 3；如果选择了 3，那么他的下一层的选择列表只有 1 和 2,那么这个时候就要引入一个 used 数组来记录使用过的数字；算法签名如下：

```
void backtrack(vector<int>& nums,vector<bool>&used,vector<int>& path)
                                                    //你也可以把 used 设置为全局变量
```

②找结束条件

```
        if(path.size()==nums.size()){
                res.push_back(path);
                return;
        }
```

③找准选择列表

```
for(int i=0;i<nums.size();i++)
{
    if(!used[i])//从给定的数中除去用过的，就是当前的选择列表
    {
    }
}
```

④判断是否需要剪枝

不需要剪枝，或者你可以认为，!used[i]已经是剪枝

⑤做出选择

```
for(int i=0;i<nums.size();i++)
{
    if(!used[i])//从给定的数中除去用过的，就是当前的选择列表
    {
        path.push_back(nums[i]);//做选择
        used[i]=true;//设置当前数已用
        backtrack(nums,used,path);//进入下一层
    }
}
```

⑥撤销选择

```
void backtrack(vector<int>& nums,vector<bool>&used,vector<int>& path)//used 初始化为 false
{
    if(path.size()==nums.size()){
        res.push_back(path);
        return;
    }
    for(int i=0;i<nums.size();i++)//从给定的数中除去，用过的数，就是当前的选择列表
```

```
    {
        if(!used[i]){   //如果没用过
            path.push_back(nums[i]);//做选择
            used[i]=true;//设置当前数已用
            backtrack(nums,used,path);//进入下一层
            used[i]=false;//撤销选择
            path.pop_back();//撤销选择
        }
    }
}
```

总结：可以发现“**排列**”类型问题和“**子集、**组合”问题不同在于：“**排列**”问题使用 used **数**组来标识选择列表，而“**子集、**组合”问题则使用 start **参数**

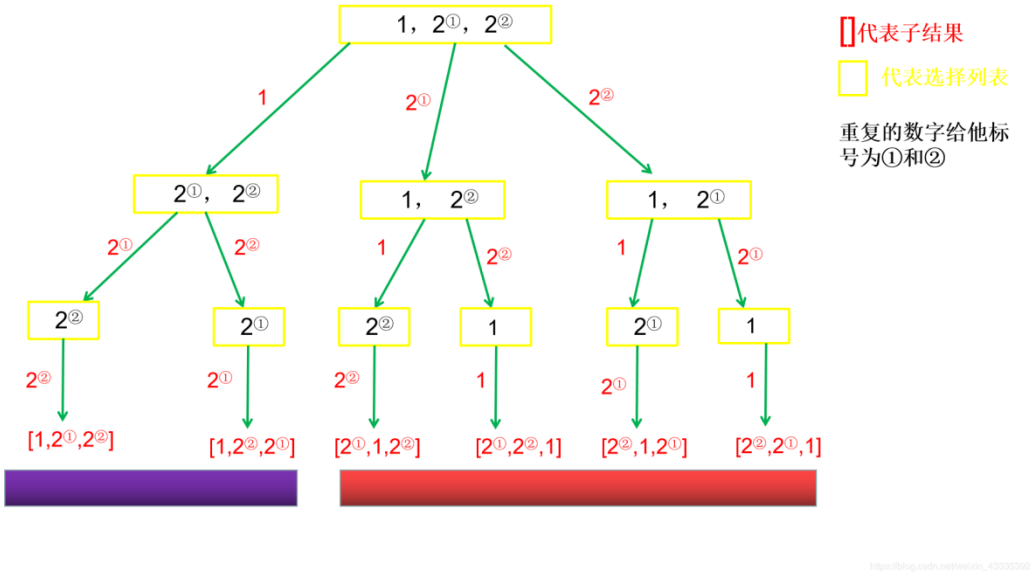
B. 全排列 II(剪枝思想)--问题描述

给定一个**可包含重复数字的序列，返回所有不重复的全排列。**

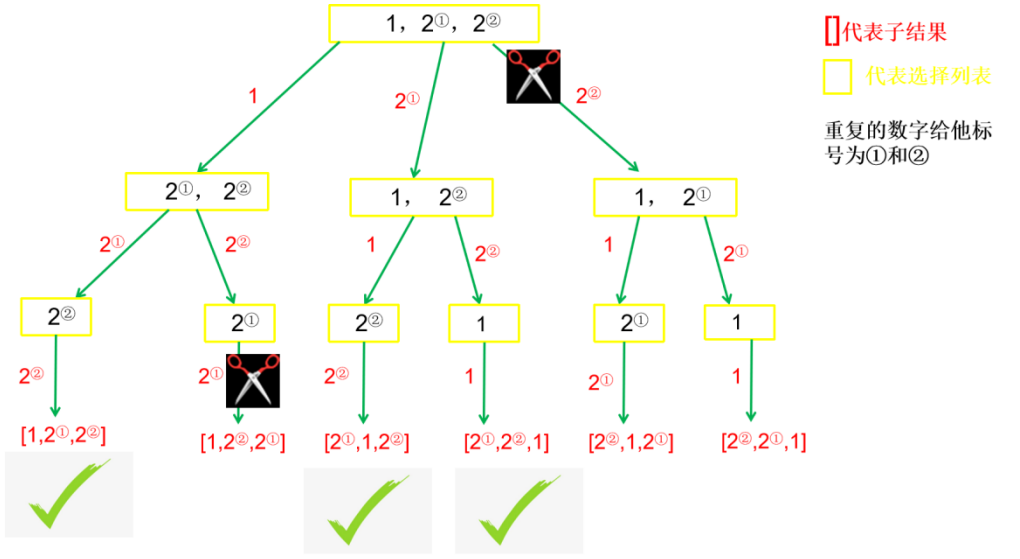
当遇到有重复元素求子集时，先对 nums 数组的元素排序，再用 if(i>start&&nums[i]==nums[i-1])来判断是否剪枝，那么在排列问题中又该怎么做呢？

①递归树

依旧要画递归树，判断在哪里剪枝。这个判断不是凭空想出来，而是看树上**的重复部分，而**归纳出来的：



可以看到，有两组是各自重复的，那么应该剪去哪条分支？首先要弄懂，重复结果是怎么来的，比如最后边的分支，选了第二个 2 后，,竟然还能选第一个 2，从而导致最右边整条分支都是重复的



②③不再赘述，直接看④

④判断是否需要剪枝，如何编码

有了前面“子集、组合”问题的判重经验，同样首先要对题目中给出的 nums 数组排序，让重复的元素并列排在一起，在 if(i>start&&nums[i]==nums[i-1])，基础上修改为 if(i>0&&nums[i]==nums[i-1]&&!used[i-1])，语义为：当 i 可以选第一个元素之后的元素时(因为如果 i=0，即只有一个元素，哪来的重复？有重复即说明起码有两个元素或以上,i>0)，然后判断当前元素是否和上一个元素相同？如果相同，再判断上一个元素是否能用？如果三个条件都满足，那么该分支一定是重复的，应该剪去

```
void backtrack(vector<int>& nums,vector<bool>&used,vector<int>& path){
    //used 初始化全为 false
    if(path.size()==nums.size()){
        res.push_back(path);
        return;
    }
    for(int i=0;i<nums.size();i++)//从给定的数中除去，用过的数，就是当前的选择列表
    {
        if(!used[i])
        {
            if(i>0&&nums[i]==nums[i-1]&&!used[i-1])//剪枝，三个条件
                continue;
            path.push_back(nums[i]);//做选择
            used[i]=true;//设置当前数已用
            backtrack(nums,used,path);//进入下一层
            used[i]=false;//撤销选择
            path.pop_back();//撤销选择
        }
    }
}
```

C. 字符串的全排列--问题描述(剪枝思想)

输入一个字符串，打印出该字符串中字符的所有排列。你可以以任意顺序返回这个字符串数组，**但里面不能有重复元素**。

```
//vector<string>res 为全局变量，表示最终的结果集，最后要返回的
class Solution {
public:
  void backtrack(string s,string& path,vector<bool>& used)//used 数组
```

```
    {
        if(path.size()==s.size())
        {
            res.push_back(path);
            return;
        }
        for(int i=0;i<s.size();i++)
        {
            if(!used[i])
            {
                if(i>=1&&s[i-1]==s[i]&&!used[i-1])//判重剪枝
                    continue;
                path.push_back(s[i]);
                used[i]=true;
                backtrack(s,path,used);
                used[i]=false;
                path.pop_back();
            }
        }
    }

vector<string> permutation(string s) {
        if(s.size()==0)
            return{};
        string temp="";
        sort(s.begin(),s.end());
        vector<bool>used(s.size());
        backtrack(s,temp,used);
```

```
        return res;
    }
};
```

再次总结：“排列”类型问题和“子集、组合”问题不同在于：“排列”问题使用 used 数组来标识选择列表，而“子集、组合”问题则使用 start 参数。另外还需注意两种问题的判重剪枝！！