回溯是递归的副产品,只要有递归就会有回溯

「因为回溯的本质是穷举，穷举所有可能，然后选出我们想要的答案」,如果想要高效就要在此基础上加一些剪枝的操作，但也改变不了回溯法就是穷举的本质

回溯法适用的题型：

1、组合问题：N个数**按一定规则**找出k个数的集合

2、排序问题：N个数**按一定规则**全排列，有几种排列方式

3、切割问题：一个**字符串**按**一定规则**有**几种切割方式**

4、子集问题：一个N个数的集合里有多少符合条件的子集

5、棋盘问题：**N皇后，解数独**等等

所有回溯法都可以抽象为**树形结构**

组合无序，排列有序即可

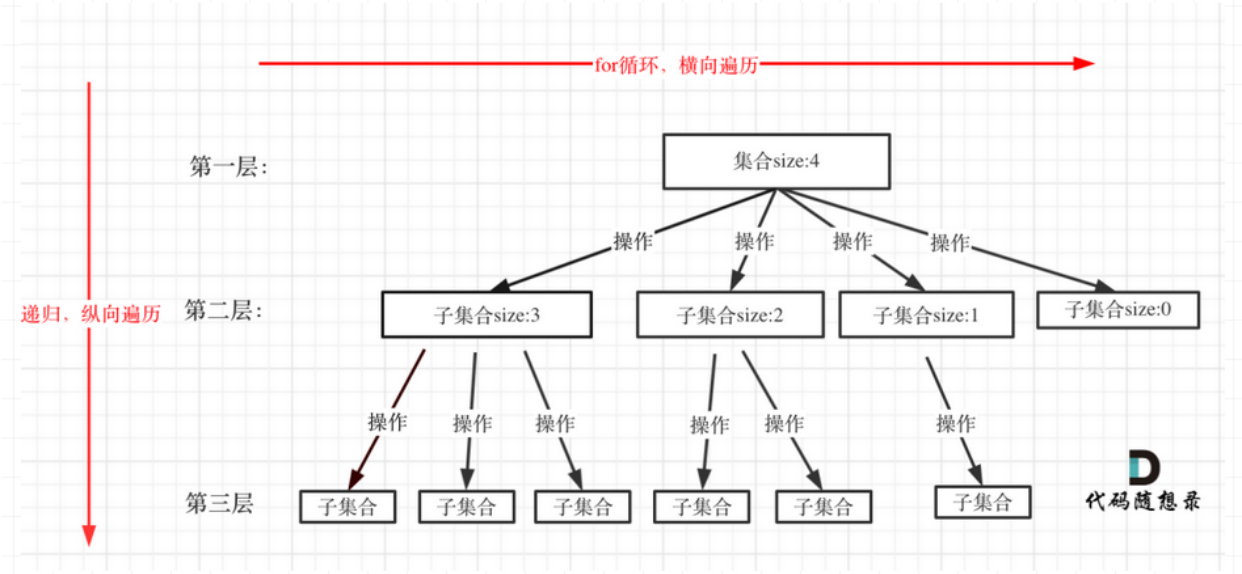
如何理解回溯法?

回溯法解决的问题都可以抽象为树形结构,

> 因为**回溯法**解决的都是**在集合中递归查找子集，集合的大小构成了树的深度**
> **step1:**
> **回溯函数模板的返回值以及参数**
> **回溯算法模板返回值一般为void,**
> **step2:**
> **回溯算法的终止条件**
> **一般来说搜到叶子结点时，也就找到了满足条件的一条答案，把这个答案存放起来，并结束本层递归**

```
void backtracking(参数){
 if(终止条件){
 存放结果;
 return ;
 }
}
```

如图：



注意图中，我特意举例集合大小和孩子的数量是相等的！

core思想就是：for循环**横向遍历**，递归**纵向遍历**

step3:回溯算法的单层逻辑

回溯函数的遍历过程伪代码如下：

```
for(选择:本层集合中元素(树中结点孩子的数量就是集合大小)){
 处理结点;
 backtracking(路径，选择列表);//递归
 回溯,撤销处理结果
}
```