回溯算法基础

何为回溯法?

回溯法又称回溯搜索法，是一种搜索的方式。

回溯是递归的副产品，只要有递归就会有回溯。

回溯法的效率

回溯法并不是什么高效的算法

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。

为让回溯法高效一点，可以加一些剪枝的操作。

适用问题

回溯法，一般可以解决以下几类问题：

- 组合问题：N个数里面按一定规则找出k个数的集合
- 切割问题：一个字符串按一定规则有几种切割方式
- 子集问题：一个N个数的集合里有多少符合条件的子集
- 排列问题：N个数按一定规则全排列，有几种排列方式
- 棋盘问题：N皇后，解数独等等

注：**组合是不强调元素顺序的，排列是强调元素顺序。**

组合无序，排列有序。

例如：{1, 2} 和 {2, 1} 在组合上，就是一个集合，因为不强调顺序，而要是排列的话，{1, 2} 和 {2, 1} 就是两个集合了。

理解回溯法

回溯法解决的问题都可以抽象为树形结构！所有的都是！

因为回溯法解决的问题大多都是在集合中递归查找子集，集合的大小就构成了树的宽度，递归的深度，都构成的树的深度。

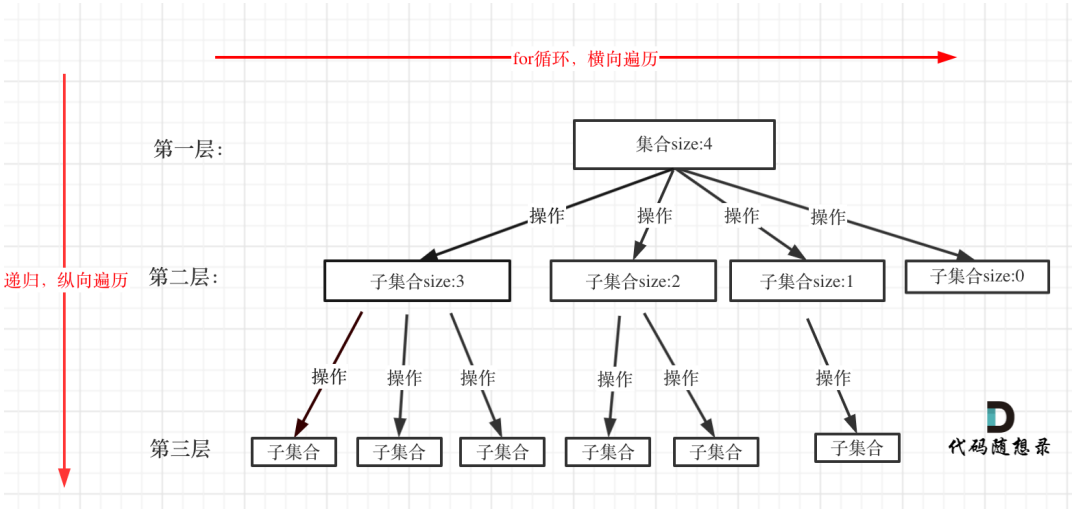
递归就要有终止条件，因此必然是一棵高度有限的树 N叉树

回溯法模板

回溯三部曲：

- 回溯函数模板返回值及其参数
  - 在回溯算法中，按个人习惯为函数起名字为backtracking，这个起名随意
  - 回溯算法中函数返回值一般为void。
  - 参数，因为回溯算法需要的参数可不像二叉树递归的时候那么容易一次性确定下来，所以一般是先写逻辑，然后需要什么参数，就填什么参数。
  - ```
    void backtracking(参数)
    ```
    ```

- 回溯函数终止条件
  - 般来说搜到叶子节点了，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。
  - ```
    if (终止条件) {
        存放结果;
        return;
    }
    ```

- 回溯搜索的遍历过程
  - 回溯法一般是在集合中递归搜索，集合的大小构成了树的宽度，递归的深度构成的树的深度。
  - 

  - 图中，特意举例集合大小和孩子的数量是相等的！
  - ```
    for (选择：本层集合中元素（树中节点孩子的数量就是集合的大小）) {
        处理节点;
        backtracking(路径，选择列表); // 递归
        回溯，撤销处理结果
    }
    ```
  - for循环就是遍历集合区间，可以理解一个节点有多少个孩子，这个for循环就执行多少次
  - backtracking这里自己调用自己，实现递归。
  - 可以从图中看出**for循环可以理解是横向遍历，backtracking（递归）就是纵向遍历**，这样就把这棵树全遍历完了，一般来说，搜索叶子节点就是找的其中一个结果了。

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

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

小结

回溯和递归是相辅相成的。

回溯法其实就是暴力查找，并不是什么高效的算法

回溯法可以解决几类问题，可以看出每一类问题都不简单。

回溯法解决的问题都可以抽象为树形结构（N叉树），并给出了回溯法的模板。