**力扣刷题整理**

1. **数组**

**704二分查找等于某个值的元素**

注意查找的是左闭右闭区间还是左闭右开区间，如果是左闭右闭那么循环的条件是

for left <= right

left = mid + 1 或 right = mid - 1

由于每次查找都会将查找范围缩小一半，因此二分查找的时间复杂度是 O(log n) log n < n

伪码：

if len(nums)==0 then return -1

left=0

rught=len(nums)-1

for left<=right {

mid = (left+right)/2

if target<mid right=mid-1

if target>mid left=mid+1

else return mid

**27移除等于某个值的全部元素**

用双指针，前一个指针判断是否等要移除的值，后一个指针是修改后数组的长度（后指针，前指针--->）

前一个指针遍历完一遍整个数组，用for range

后一个指针用来指向需要修改的位置

伪码：

p:=0

for range nums

if v1=nums then nums[p]=v p++

**[977有序数组的平方](https://leetcode.cn/problems/squares-of-a-sorted-array/)后排序(平方前有正负数）**

利用双指针，一个指头，一个指尾

两指针所指的数平方后比较，谁大谁从后向前先加入结果数组，指针移一位

1. **字符串**

**1047删除字符串中的所有相邻重复项**

把字符切片当作栈来使用

逐步遍历字符串的字符，与字符切片的最后一个元素（栈顶）一样的则从字符切片里删除这个元素，依次遍历完字符串

1. **链表**

**24两辆交换链表中的节点**

cur=dummyHead;如果cur.Next!=nil&&cur.Next.Next!=nil 利用cur交换cur后面的两节点（需设置临时指针）；然后cur往后移两位再做相同的事

**206反转链表**

需要三个指针：head->pre1->pre2

循环：pre1.Next=head;head=pre1;pre1=pre2;pre2=pre2.Next

1. **二叉树**

二叉树理论基础

**种类**

满二叉树：根和节点都满

完全二叉树：最后一层不一定满，其它层都满

二叉搜索树：所有节点满足：左小于中小于右

平衡二叉搜索树：左子树和右子树的高度差不能超过1

红黑树：自平衡二叉搜索树

**存储**

链式存储：左右指针

线性存储：i的左孩子：2i+1，右孩子：2i+2

**遍历方式**

1. 深度优先搜索DFS（前中后序遍历），一般使用递归法和迭代法（使用栈，先不学）
2. 广度优先搜索BFS（层序），迭代法（使用队列）

代码随想录网站515题还没有看完

1. **递归**

写递归题三部曲：

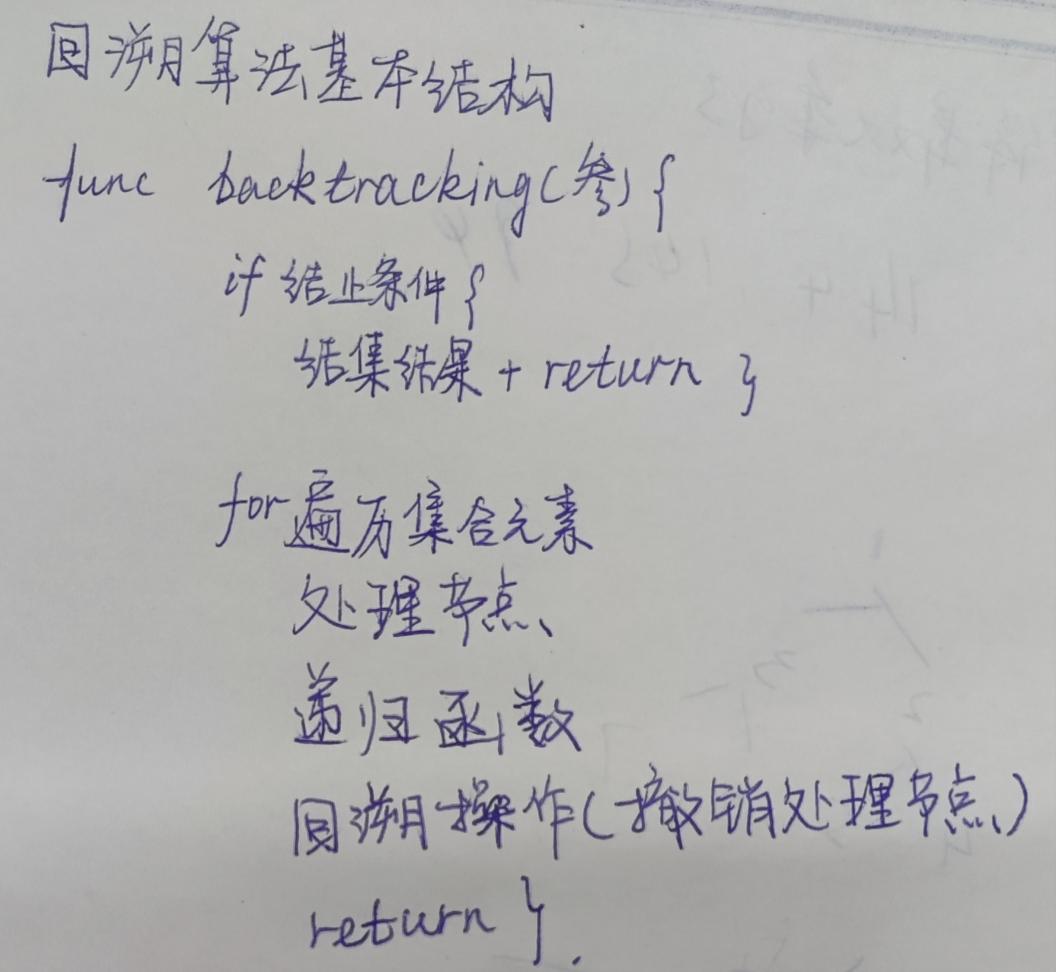
**a确定递归的参数和返回值**

**b确定终止条件（一般是遍历到null返回）**

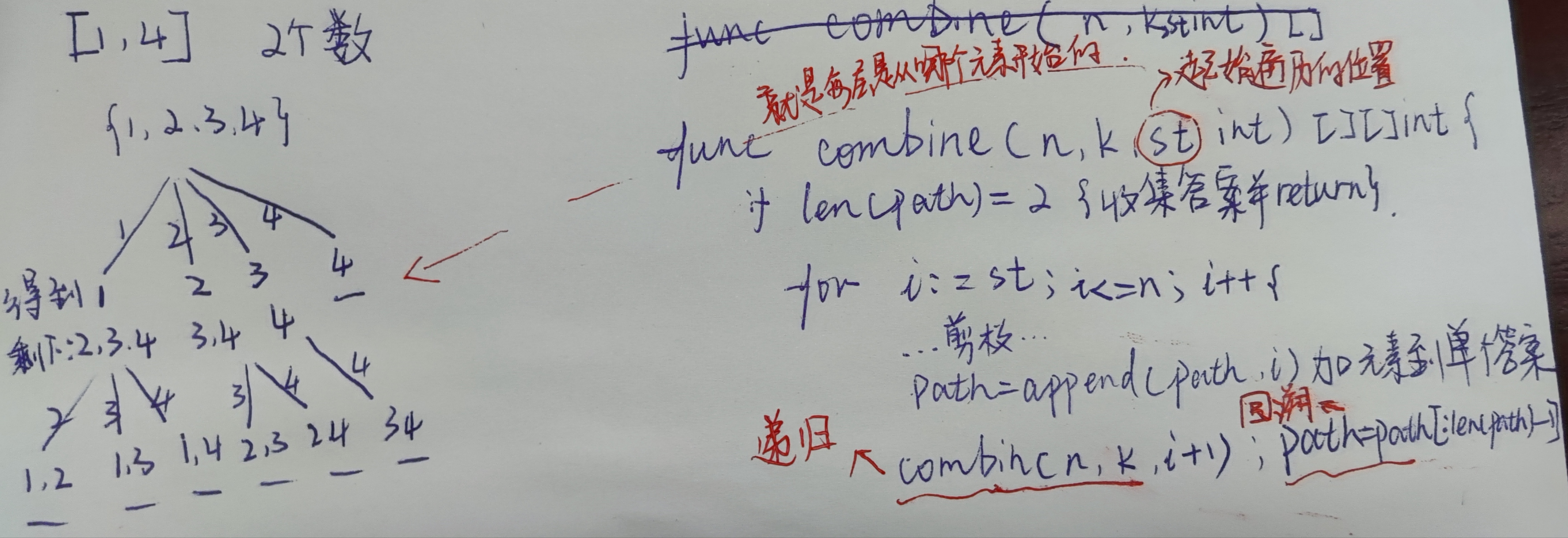
**c确定单层递归的逻辑（前中后）**

1. **回溯算法**
2. 回溯就是递归终止的过程，可以模拟为在n叉树上操作
3. 解决：组合（无序）、排列（有序）、切割、子集、棋盘（N皇后和解数独）
4. 回溯算法一般名为backtracking（），参数可以在写函数逻辑的时候再看需要哪些参数
5. 一定会有终止条件，一般在此处收集结果之后return

回溯算法的一般结构：



77（**要求会默写**）组合：给定两个整数 n 和 k，返回范围 [1, n] 中所有可能的 k 个数的组合，顺序任意。



1. **贪心**

局部最优

每次都选一个最大的，或最好的

结果是否最优：只能推理一下，本问题的局部最优是否能退出全局最优

1. **动态规划**

背包问题、打家劫舍、股票问题、子序列问题

关键步骤：

1. **搞清楚dp数组及下标的含义**
2. **递推公式**
3. **dp数组如何初始化**
4. **遍历顺序**
5. **打印dp数组**

**509斐波那契数列：**

初始化：dp[0]=0,dp[1]=1

第n个数dp[n] = dp[n-2]+ dp[n-1]

**70爬楼梯（每次只能爬1或2阶）**

初始化：dp[1]=1,dp[2]=2

第n个数dp[n] = dp[n-2]+ dp[n-1]

**746最小花费爬楼梯**

dp[i] = min(dp[i-2]+cost[i-2], dp[i-1]+cost[i-1])

**62不同路径**

初始化：

var table [][]int = make([][]int, n)

    for i := 0; i < n; i++ {

        table[i] = make([]int, m)

        table[i][0] = 1

    }

    for j := 0; j < m; j++ {

        table[0][j] = 1

    }

递推公式：

table[i][j] = table[i-1][j] + table[i][j-1]