**力扣刷题整理**

1. **查找算法**（7个）-找到一个目标值

顺序、二分（有序）、插值、斐波那契、树表、分块、哈希查找

1. **排序算法**（6个）

****

* 1. **冒泡排序：**一层循环：0<=i<n-1；二层循环:0<=j<n-i-1；第一层是需要排序的元素个数，第二层是这个元素需要比较的次数。
  2. **选择排序：**每次循环选择一个最小的数放到未排序列的最左边；优化：每次循环选择选最小的和最大的分别放在两边。
  3. **插入排序**：从后往前扫描，找到合适位置插入。
  4. **希尔排序：**是插入排序的优化版，按照一定的间隔分为若干个子序列，对每个子序列进行插入排序，逐步缩小间隔到1。
  5. **归并排序：**（分治思想）将待排序数组拆分成若干个小的子数组，然后对这些子数组进行排序，最后将这些子数组合并成一个有序的数组（用两个指针进行合并）。需要额外的空间来存放临时数组。
  6. **快速排序：**通过分治策略和递归方法的结合实现的。先选出一个基准元素，将待排序的字段分成一部分比基准元素小，一部分比基准元素大。然后对这两部分一直递归下去完成相同的操作，直到分成的那部分只有一个元素再返回。虽然快速排序是一种原地排序算法，但是在递归过程中需要使用栈来保存递归信息，所以空间复杂度比较高。时间复杂度是O(nlogn)-O(n^2)。空间复杂度是O(logn)-O(n)。
  7. **堆排序：**  把数据建堆（完全二叉树），根节点和最后叶子节点互换后再次建堆...最好和最坏的时间复杂度都一样。常数因子（建堆等花销）较大。

**小规模数据时可以用冒泡排序、选择排序和插入排序他们实现简单但是时间复杂度大。中规模数据时可以选择希尔排序，它是优化版的插入排序，时间复杂度比较小。大规模数据排序且内存足够大时可以用归并排序，他读取次数比较少且稳定。内存不够且乱序时选用快速排序。基本有序时选用堆排序。**

1. **切片与数组**

**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

**242判断字符串 s 和 t 中每个字符出现的次数是否相同**

s=[]byte(s);t...； 对字符切片进行排序

sort.Slice(s,func(i,j int)bool{return s[i]<s[j]});(t一样）；return string(s)==..

**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**

输出一个字符byte/rune是输出对应的码值(A~Z:65~90;a~z:97~122)，如果需要输出字符需把它转为string格式：string(‘a’),字符串则照原样输出

源码：type byte uint8;type rune uint32;rune一般用来存放大码值字符

**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. **哈希表（Map）**

Go里的Map需make之后才能存数据，且不可指定容量，不存在的key对应的vlue为零值；（v,ok:=... ok为false）

**1数组内两数之和等于target**

只需遍历一遍数组，然后判断：p, ok := hash[target-v]；如果ok为false说明没有target-v这个值，我们存入当前v：m[v]=i;ok为true则返回i和p

**349两个数组的交集**

遍历数组1得的key对应map的value如果是0则设为1（避免1有重复值）；

遍历数组2得的key对应map的value如果是1则收集（避免2有重复值）；

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（排序）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（排序）树，即使顺序输入也自动变为平衡二叉搜素树

*B树和B+树（在MySQL的索引那里）*

**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：

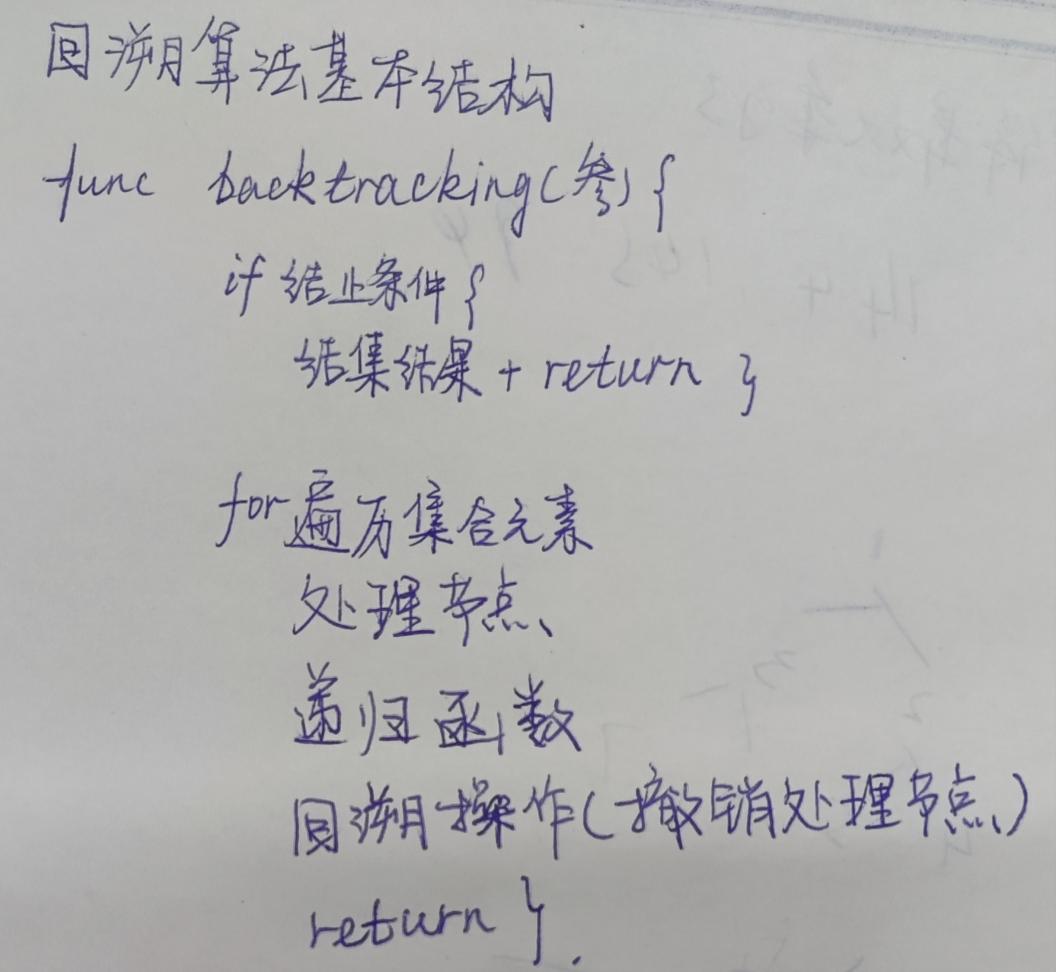
**a确定递归的参数和返回值**

**b确定终止条件（一般是遍历到null返回）**

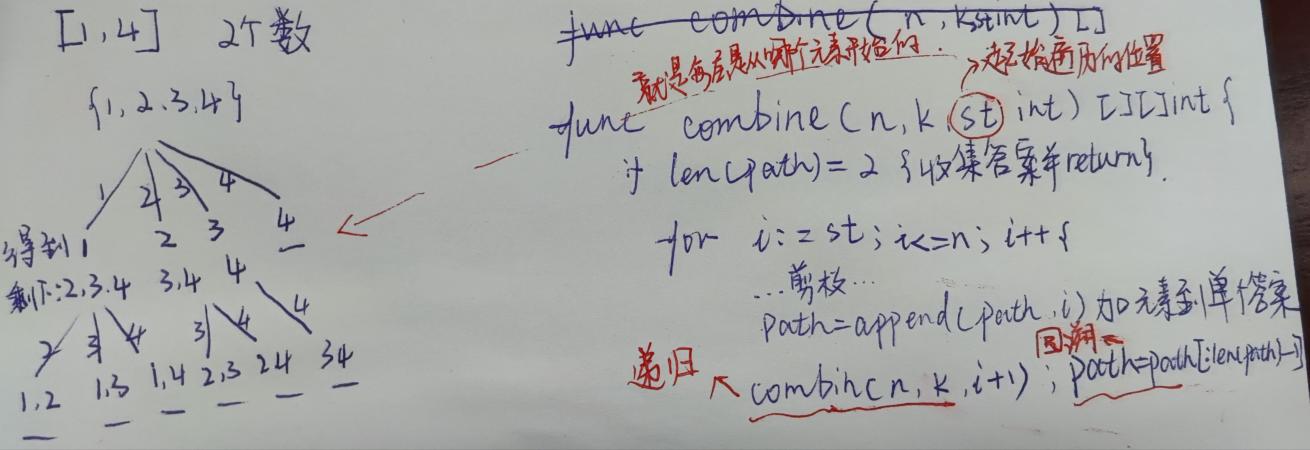
**c确定单层递归的逻辑（前中后）**

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]

**343整数拆分**

dp[i]是对i拆分得到的最大乘积数;dp[0],dp[1]=0;dp[2]=1

for i:=3;i<=n;i++{ for j:=1;j<i;j++{ dp[i]=max(dp[i],max(j\*(i-j),j\*dp[i-j])) } }

**96不同的二叉搜索树**

思路：等于左子树的种类\*右子树的种类；子树之下再分为左右子树

dp[i]表示从1到i为节点的不同二叉搜索树的和；初始化：dp[0]=1

  for i:=1;i<=n;i++{for j:=0;j<i;j++{dp[i]=dp[i]+dp[j]\*dp[i-j-1]}}

**198打家劫舍**

dp[i]是指考虑到i家以及之前的能得到的最多钱，则dp[i]有偷i和不偷i两种情况，选最大的那种：dp[i]=max(dp[i-1],dp[i-2]+nums[i])

初始化：dp[0]=nums[0];dp[1]=max(nums[0],nums[1])

**121买卖股票的最佳时机**