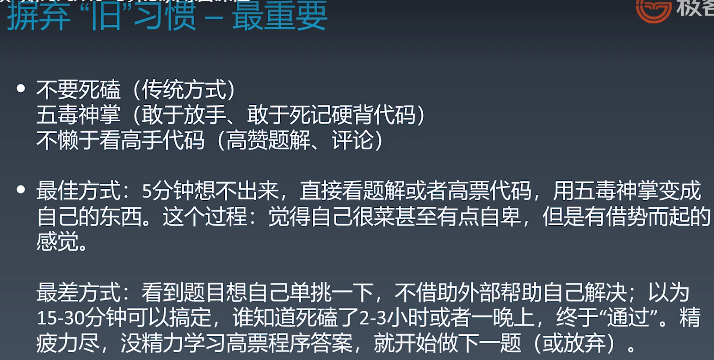
# **前言**

* 高效学习方法：五毒神掌法

五遍刷题法，强调练习算法题要过遍数，算法题不仅是做一遍通过而已，而是要通过**刻意练习**达到对算法熟练的效果

* + 第一遍：5-10分钟如果没有思路，就去题解里面按照高赞排序找最佳题解看，记住方法和代码
  + 第二遍：自己写代码
  + 第三遍：一天之后重做
  + 第四遍：一周之后重做
  + 第五遍：面试前一周重做
* 做题误区：希望一遍理解 + 做题只做一遍
* 看题之后没思路的时候：
  + 暴力解题法
  + 分析基本情况
  + 找最近重复子问题（climbstairs问题）
* 算法到最后的3种基本情况：
  + if…else…
  + for (while) loop
  + recursion
* 时间复杂度、空间复杂度：
  + 优化代码的时间复杂度方法：空间换时间/升维
* 循环遍历基本功/从两边向中间遍历基本功



# **Ww01**

## **数组**

## **链表**

## **跳表**

## **栈**

## **队列**

## **优先队列**

## **双端队列**

## **实战**

* 3sum问题-数组类型问题-双指针法
* linklist类型问题：
  + 方法固定，没有太多算法考点，主要是next/pre指针操作，需要多练习熟练掌握；
  + 唯一例外：环形链表问题
* 环形链表问题的3种思路：暴力法/hash表/快慢指针
* 快慢指针法在链表问题中很常见
* 什么样的问题可以用栈来解决：有最近相关性的问题=》剥洋葱型成对的问题
* 最小括号问题：
* 暴力法：将有效括号replace为空
* 栈
* 最小栈问题：
  + 辅助栈法（使用2个栈：一个维护出入关系， 一个维护最小栈）
* 用2个栈/队列可以解决的问题： 用栈实现队列， 用队列实现栈
* 柱状图中的最大矩形问题：
  + 暴力法
  + stack
* 滑动窗口问题： 用队列解决

# **Ww02**

## **集合**

## **哈希表**

## **映射**

## **树**

## **二叉树**

## **二叉搜索树**

## **堆**

## **二叉堆**

## **图**

# **Ww03**

## **泛型递归**

* 递归的实现+特性
  + 递归本质是循环，是不断调用函数本身的循环，向下进入到下一层，向上回到原来一层；
  + 每一层的环境和数据都是一份拷贝（是局部的，不会互相影响），主角穿越到不同层中（函数的参数，会发生变化）
* 递归的3个思维要点
  + 不要人肉递归（最大误区）
  + 找到最近最简方法，将其拆解成可重复解决子问题（最小重复子问题）
  + 数学归纳法
* 递归最典型 的例子：Fibonacci数列和n!
* Python代码模板

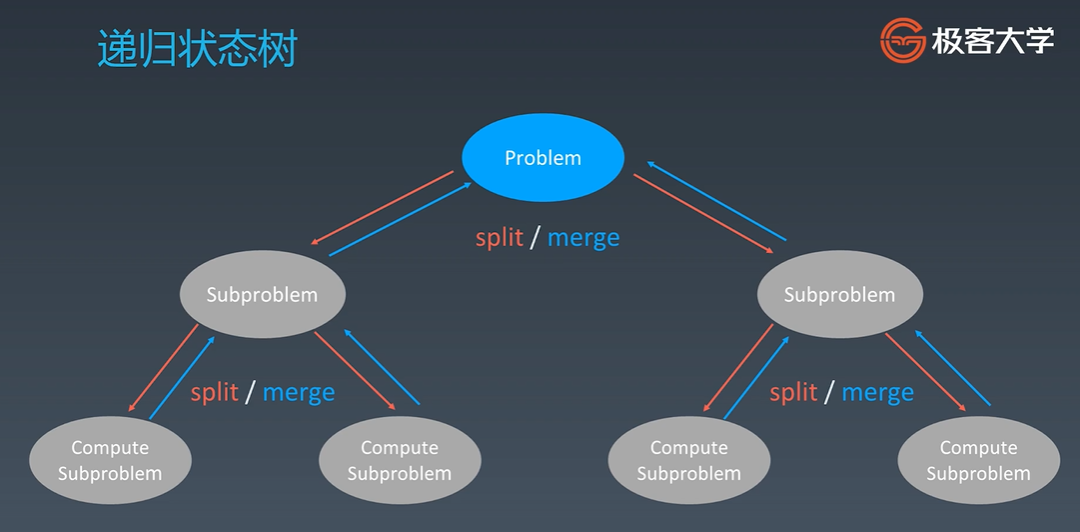


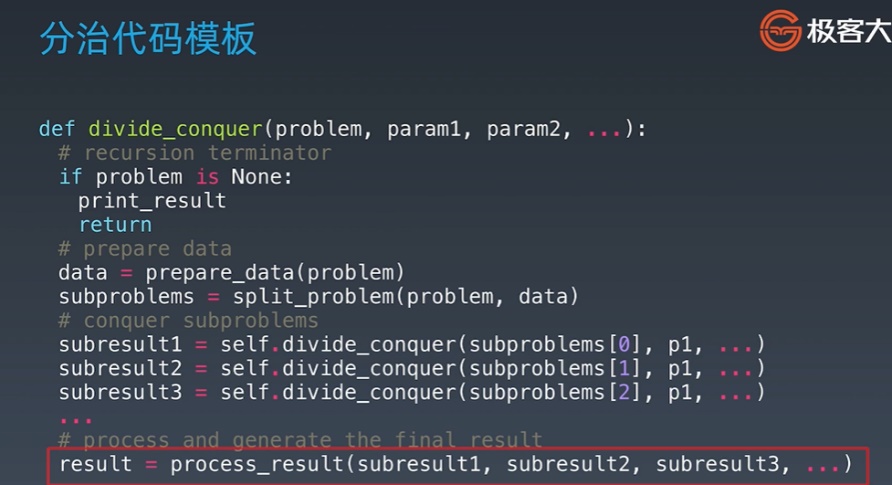
* 实战
* 爬楼梯问题：
  + 根据给出条件反向推导出递归公式：只能走1步或2步
  + 当n时，如果站在n-1上，走一步到n；如果站在n-2上，直接走2步到n；
  + F(n) = f(n-1) + f(n-2)
  + 考虑会不会有漏/重复
* 括号生成问题：
  + 可以先把所有情况生成出来，然后输出之前筛选有效的括号
  + 生成的时候就加判断条件：1)left未超标就可以放左括号 2)left >= right才可以放右括号
  + 括号生成问题给后面很多问题提供了一个思路，就是给出n个格子，这个格子里面可以放左括号，也可以放右括号

## **树的递归**

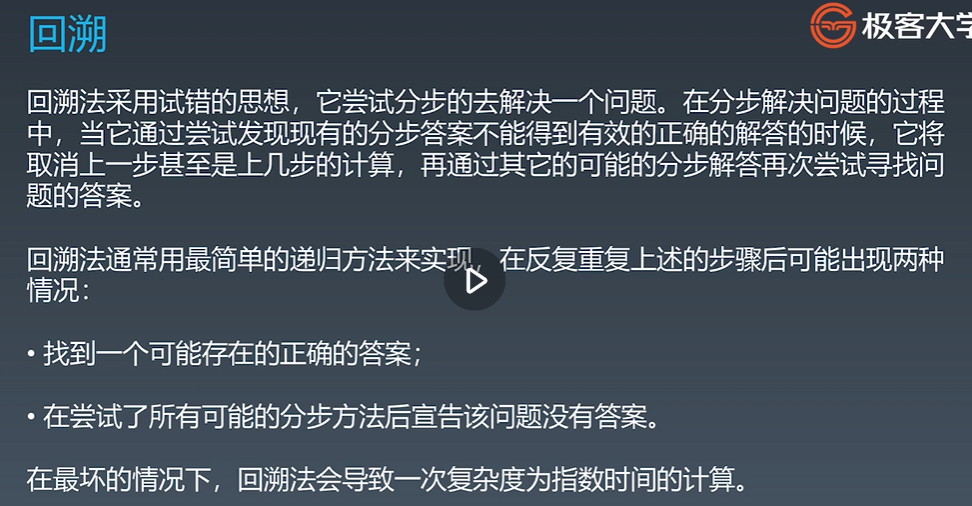
* 判断二叉搜索树问题
  + 递归（不能只递归比较左右结点，要递归左子树和右子树）
  + 中序遍历是递增数列，遍历过程中判断当前元素是否大于等于前一个元素即可

## **分治(divide and conquer)**

* 本质上是递归，特殊的递归
* 遇到一个问题找它的重复性:
  + 最优重复性->动态规划；
  + 最近重复性->一般的递归，分治，回溯
* 分治算法的思想
* 分治代码模板



## **回溯(backtracking)**



* 最典型的应用是处理八皇后问题和数组问题
* 实战
  + Pow(x, n)
    - 暴力法：for循环 O(n)
    - 分治法： O(logN)
  + 子集问题：
    - 分治法
    - 迭代法\*
  + 众数
  + 电话号码的字母组合-分治
  + N皇后问题-分治

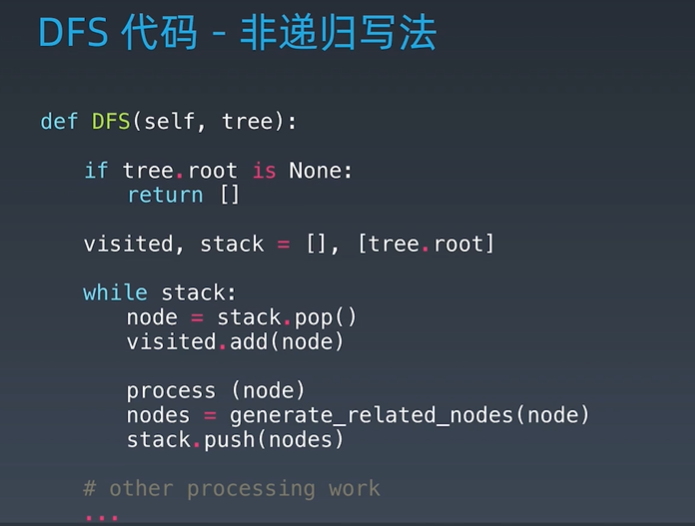
# **Ww04**

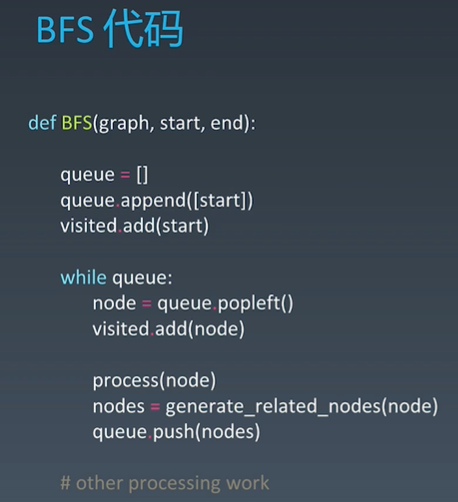
## **深度优先搜索和广度优先搜索**

* 搜索的本质就是把所有的结点都遍历一次且只遍历一次，按照遍历顺序不同分为深度优先搜索和广度优先搜索，实现方式有递归 和 非递归两种方式：
  + 递归： 代码比较简单
  + 非递归：栈/队列 + for loop
* 深度优先搜索：
  + 递归
  + 非递归-手动维护一个栈
* 广度优先搜索：非递归-手动维护一个队列

**程序模板要写的滚瓜烂熟**

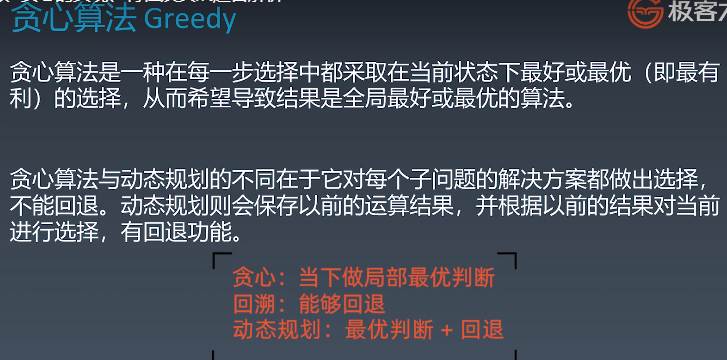
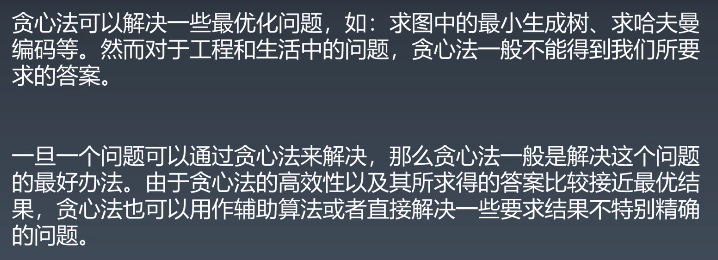
* 实战：
  + 找重复性 -最小重复子问题
  + 二叉树的层序遍历

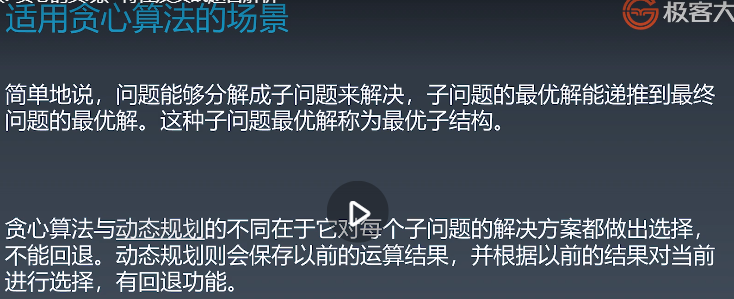
 



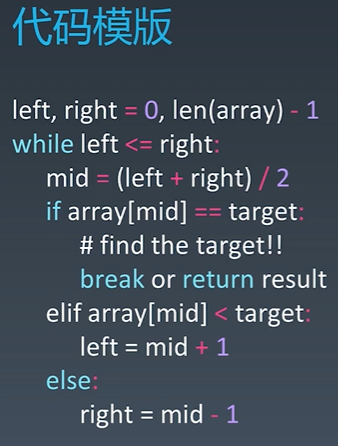
## **贪心算法**

* 每一步都采取局部最优，以期望达到全局最优；
* 适用于贪心法的情况：有特殊性/能够证明用最简单粗暴的贪心法可以得到最优解，局部最优可以导致全局最优
* 贪心算法的**难点**是怎样证明贪心算法是最优的
* 贪心算法可以从前面开始，也可以从局部开始贪心，也可以从后往前



## **二分查找**

* 使用二分查找的前提是：
  + 数组具有单调性monotonicity
  + 数组有界 bounded
  + 能够通过索引进行访问 index accessible
* 每次找数组的中值，通过对比target和中值大小缩小搜索范围，从而减小时间复杂度 **思维逻辑 代码能力 debug能力**

# **Ww05**

## **递归感触**

* 拒绝人肉递归：低效+很累
* 找到最近最简方法，将其拆解成可重复解决的问题
* 数学归纳思维（抵制人肉递归的诱惑）
* 画递归状态树\*\*

## **动态规划得实现及关键点 dynamic programming**

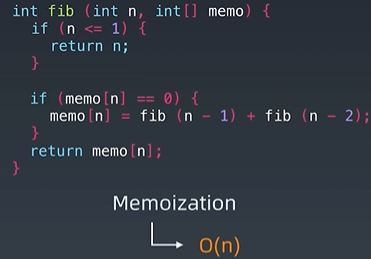
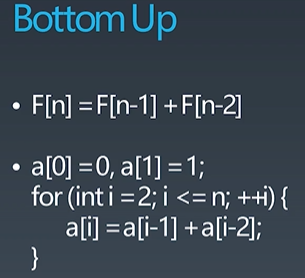
* **理解**：divide & conquer + optimal substructure分治 + 最优子结构
* **关键点**（3句话）：
  + 动态规划和递归或者分治没有根本上的区别（关键看有无最优子结构）
  + 共性：找到重复子问题
  + 差异性：最优子结构，中途可以淘汰次优解

\*所谓最优子结构：最小值是..最多是..最快…

* **对时间复杂度的影响**
  + 傻递归：一般指数级
  + 动态规划：一般是多项式级别甚至线性级别
* **编码**(2种思维方式)
  + 自顶向下：递归 + 记忆化搜索 recursive + memoize
  + 自下向上递推bottom up： 递推 + for loop

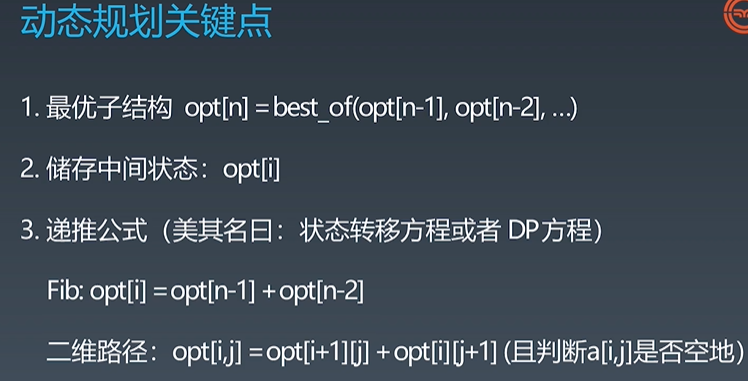
## **Fibonacci数列（一维）**

* 傻递归：时间复杂度是指数级的 2^n
* 动态规划：线性时间复杂度O(n )

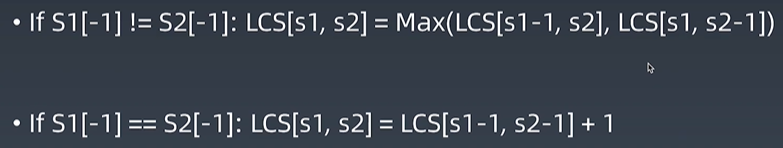
## **Count the paths（二维）**

* 最小重复子问题：右点到终点 / 下点到终点
* 最优子结构：
  + paths(start, end) = paths[A, end+ paths[B, end]
  + 不同问题种的最优子结构不同，可能是取最大值，可能是取最小值，这里是二者之和
* **关键**是找对**递推公式** ->使用数学归纳法，从最简单的情况出发（递归终止条件，递推初始值）
* **建议使用递推的思维方式**



## **最长公共子序列（2个字符串）**

* 2个字符串构建一个二维数组 -> 思维转移
* 递推公式



## **动态规划思维小结**

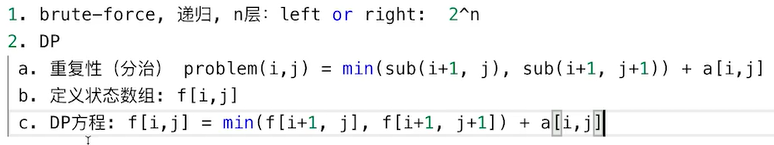
* 打破自己的思维惯性，形成机器思维->找重复性（机器只会if..else/for loop/recursive）
* 理解复杂逻辑的关键

## **爬楼梯问题的进阶**

* 初始问题1,2
* 1,2,3
* 相邻两步的步伐不能相同

## **三角形最小路径和**

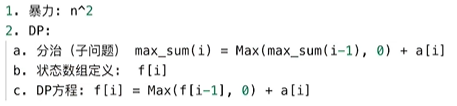
* Brute-force，递归， n层：left or right: 2^n
* DP
* 重复性（分治）
* 定义状态数组（OPT应该是什么样的）
* DP方程



Python中add LRU\_cache ?

## **子序列的最大和**

* 提高自己认知能力的时候：找到自相似性办法，即重复性办法，能够化繁为简，保证逻辑上简洁+严谨可证明的；
* **认知误区**：大概看一下，好像可以是这样的，数学上不严谨，逻辑上没有自相似性；
* 暴力： n^2
* DP
  + 分治(子问题)
  + 最大子序和 = 当前元素自身最大 , 或者包含之前后最大
  + 状态数组定义
  + DP方程

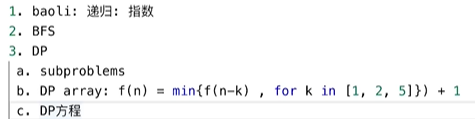


## **子序列的最大乘积**

把正的最大值保存下来，把负的最大值也保存下来

## **零钱兑换**

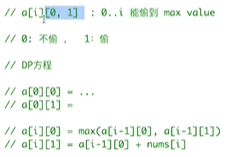
* 暴力：递归，指数级
* BFS
* DP

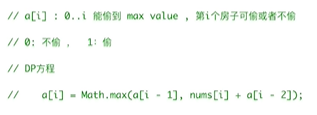


## **子序列的最大集**

## **打家结舍**

2种方法





## **打家劫舍2**