学习算法过程中的语言问题

前置知识：掌握一门语言

一定要熟练掌握一门编程语言，任何语言都可以

一份java代码可以用gpt-4来转化的实例

学习算法过程中的语言问题

开通gpt-4的最方便的途径(截止2023年)简单介绍
1，某宝上购买一整套服务(省心但是稍微贵一些，贵不太多)
2，自己手动(利用IOS升级GPT-4)
    0，已经拥有了gpt的账号
    A，IOS美区账号
    B，美区账号充值，礼品卡，某宝
    C，IOS美区账号，chatpgt app
    D，利用IOS美区账号的钱，在手机里升级到gpt-4
    E，手机端升级，网页端同步升级

注意 ： 最好不要使用虚拟信用卡的方式，容易被封

左程云

学习算法入门提醒

前置知识：掌握一门语言

先来个有意思的实验

一开始有100个人，每个人都有100元
在每一轮都做如下的事情 ：
每个人都必须拿出1元钱给除自己以外的其他人，给谁完全随机
如果某个人在这一轮的钱数为0，那么他可以不给，但是可以接收
发生很多很多轮之后，这100人的社会财富分布很均匀吗？

既然是程序员，当然用代码做实验啊

左程云

学习算法入门提醒

1，善于、乐于折腾。勤写代码找到乐趣，只看课没有用。看懂了一定要确保自己手写正确，再继续下个内容

2，算法的学习节奏，不像大学，像高中。区别是一个为了应试，一个在磨练手艺，代码人就是手艺人

3，关于复习，尽量冲击到一定的题量再整体复习，不要频繁复习，会拉长周期，而且很多是无效的复习

左程云

二进制和位运算

前置知识：无
建议：不要跳过

1）二进制和位的概念
2）正数怎么用二进制表达
3）负数怎么用二进制表达
4）打印二进制；直接定义二进制、十六进制的变量
5）常见的位运算（|、&、^、~、<<、>>、>>>）
6）解释打印二进制的函数
7）注意|、&是位运算或、位运算与；||、&&是逻辑或、逻辑与，两者是有区别的
8）相反数
9）整数最小值的特殊性(取绝对值还是自己)
10）为什么这么设计二进制（为了加法的逻辑是一套逻辑，没有条件转移），那么为啥加法逻辑如此重要呢？
11）关于溢出（自己确保自己的调用所得到的结果不会溢出，一定是自己确保的，计算机不会给你做检查）
12）位运算玩法很多很多，特别是异或运算（后面的课会详细讲述）、如何用位运算实现加减乘除（后面的课会详细讲述）

左程云

选择、冒泡、插入排序

前置知识：无
建议：会的同学直接跳过

本节内容：

选择排序一句话：i~n-1范围上，找到最小值并放在i位置，然后i+1~n-1范围上继续

冒泡排序一句话：0~i范围上，相邻位置较大的数滚下去，最大值最终来到i位置，然后0~i-1范围上继续

插入排序一句话：0~i范围上已经有序，新来的数从右到左滑到不再小的位置插入，然后继续

左程云

对数器

建议：不要跳过，非常重要的自我验证技巧

对数器的试用场景

你在网上找到了某个公司的面试题，你想了好久，感觉自己会做，但是你找不到在线测试，你好心烦..

你和朋友交流面试题，你想了好久，感觉自己会做，但是你找不到在线测试，你好心烦..

你在网上做笔试，但是前几个测试用例都过了，突然一个巨大无比数据量来了，结果你的代码报错了，如此大的数据量根本看不出哪错了，甚至有的根本不提示哪个例子错了，怎么debug？你好心烦…

对数器

对数器的实现

1，你想要测的方法a（最优解）
2，实现复杂度不好但是容易实现的方法b（暴力解）
3，实现一个随机样本产生器（长度也随机、值也随机）
4，把方法a和方法b跑相同的输入样本，看看得到的结果是否一样
5，如果有一个随机样本使得比对结果不一致，打印这个出错的样本进行人工干预，改对方法a和方法b
6，当样本数量很多时比对测试依然正确，可以确定方法a（最优解）已经正确。

关键是第5步，找到一个数据量小的错误样本，便于你去带入debug
然后把错误例子带入代码一步一步排查
Print大法、断点技术都可以
对数器的门槛其实是比较高的，因为往往需要在两种不同思路下实现功能相同的两个方法，暴力一个、想象中的最优解是另一个。
以后的很多题目都会用到对数器，几乎可以验证任何方法，尤其在验证贪心、观察规律方面很有用
到时候会丰富很多对数器的实战用法，这里只是一个简单易懂的示例

二分搜索

前置知识：无
建议：1）会的同学可以跳过，2、3、4）不要跳过

1）在有序数组中确定num存在还是不存在
2）在有序数组中找>=num的最左位置
3）在有序数组中找<=num的最右位置
4）二分搜索不一定发生在有序数组上（比如寻找峰值问题）
5）“二分答案法”这个非常重要的算法，很秀很厉害，将在【必备】课程里继续

如果数组长度为n，那么二分搜索搜索次数是log n次，以2为底
下节课讲时间复杂度，二分搜索时间复杂度O(log n)

左程云

二分搜索

二分搜索不一定发生在有序数组上

峰值元素是指其值严格大于左右相邻值的元素
给你一个整数数组 nums，已知任何两个相邻的值都不相等
找到峰值元素并返回其索引
数组可能包含多个峰值，在这种情况下，返回 任何一个峰值 所在位置即可。
你可以假设 nums[-1] = nums[n] = 无穷小
你必须实现时间复杂度为 O(log n) 的算法来解决此问题。
测试链接 : https://leetcode.cn/problems/find-peak-element/

时间复杂度和空间复杂度

前置知识：选择排序、冒泡排序、插入排序、等差数列、等比数列
建议：不要跳过

1，常数操作，固定时间的操作，执行时间和数据量无关
2，时间复杂度，一个和数据量有关、只要高阶项、不要低阶项、不要常数项的操作次数表达式
举例：选择、冒泡、插入
3，严格固定流程的算法，一定强调最差情况！比如插入排序
4，算法流程上利用随机行为作为重要部分的，要看平均或者期望的时间复杂度，因为最差的时间复杂度无意义
用生成相邻值不同的数组来说明
5，算法流程上利用随机行为作为重要部分的，还有随机快速排序（【必备】课）、跳表（【扩展】课）
也只在乎平均或者期望的时间复杂度，因为最差的时间复杂度无意义
6，时间复杂度的内涵：描述算法运行时间和数据量大小的关系，而且当数据量很大很大时，这种关系相当的本质，并且排除了低阶项、常数时间的干扰
7，空间复杂度，强调额外空间；常数项时间，放弃理论分析、选择用实验来确定，因为不同常数操作的时间不同
8，什么叫最优解，先满足时间复杂度最优，然后尽量少用空间的解

左程云

时间复杂度和空间复杂度

前置知识：选择排序、冒泡排序、插入排序、等差数列
建议：不要跳过

9，时间复杂度的均摊，用动态数组的扩容来说明（等比数列、均摊的意义）
并查集、单调队列、单调栈、哈希表等结构，均有这个概念。这些内容【必备】课都会讲
10，不要用代码结构来判断时间复杂度，比如只有一个while循环的冒泡排序，其实时间复杂度O(N^2)
11，不要用代码结构来判断时间复杂度，比如：N/1 + N/2 + N/3 + … + N/N，这个流程的时间复杂度是O(N * logN)，著名的调和级数
12，时间复杂度只能是对算法流程充分理解才能分析出来，而不是简单的看代码结构！这是一个常见的错误！
甚至有些算法的实现用了多层循环嵌套，但时间复杂度是O(N)的。在【必备】课程里会经常见到
13，常见复杂度一览：
O(1) O(logN) O(N) O(N*logN) O(N^2) … O(N^k) O(2^N) … O(k^N) … O(N!)
14，时间复杂度非常重要，可以直接判断某个方法能不能通过一个题目，根据数据量猜解法，【必备】课都会讲
15，整套课会讲很多算法和数据结构，也会见到很多的时间复杂度的表达，持续看课即可

左程云

等差数列求和公式

$$S = n / 2 * (2 * a1 + (n - 1) * d)$$

其中，S 是等差数列的和；n 是项数；a1 是首项；d 是公差。

也可以认为任何等差数列的都符合：

a * n平方 + b * n + c，其中a、b、c都是常数

算法和数据结构简介

说一个我觉得比较有趣、有用的算法分类

硬计算类算法、软计算类算法
注意：这两个名词都不是计算机科学或算法中的标准术语

那为啥要提这个呢？因为很有现实意义。

硬计算类算法：精确求解。但是某些问题使用硬计算类的算法，可能会让计算的复杂度较高
大厂算法和数据结构笔试、面试题、acm比赛或者和acm形式类似的比赛，考察的都是硬计算类算法。

软计算类算法：更注重逼近解决问题，而不是精确求解。计算时间可控
比如：模糊逻辑、神经网络、进化计算、概率理论、混沌理论、支持向量机、群体智能

硬计算类算法是所有程序员岗位都会考、任何写代码的工作都会用到的。前端、后端、架构、算法所有岗位都要用到。
但是算法工程师除了掌握硬计算类的算法之外，还需要掌握软计算类的算法。

左程云

算法和数据结构简介

说一个我觉得比较宏观的数据结构分类

连续结构

跳转结构

任何数据结构都一定是这两个结构拼出来的！没有例外！

数据结构太多了，从链表、队列、栈，到可持久化线段树、树链剖分、后缀数组等等结构

后面的课都会讲到！

左程云

单双链表及其反转-堆栈诠释

前置知识：无
建议：本节内容比较初级，会的同学可以直接跳过

1）按值传递、按引用传递
（我不知道为什么如此多的同学会犯这种错误，这完全是语言问题）

2）单链表、双链表的定义

3）根据反转功能，彻底从系统角度解释链表是如何调整的

链表题目在笔试、面试中的意义就是检验coding能力如何
更难的题目会在【必备】课程里讲述

左程云

链表入门题目-合并两个有序链表

前置知识：理解链表及其基本调整
建议：做过这个题的同学跳过

将两个升序链表合并为一个新的 升序 链表并返回
新链表是通过拼接给定的两个链表的所有节点组成的

链表题目在笔试、面试中的意义就是检验coding能力如何
更难的题目会在【必备】课程里讲述

链表入门题目-两个链表相加

前置知识：理解链表及其基本调整　　建议：做过这个题的同学跳过

给你两个 非空 的链表，表示两个非负的整数
它们每位数字都是按照 逆序 的方式存储的，并且每个节点只能存储 一位 数字
请你将两个数相加，并以相同形式返回一个表示和的链表
你可以假设除了数字 0 之外，这两个数都不会以 0 开头

链表题目在笔试、面试中的意义就是检验coding能力如何
更难的题目会在【必备】课程里讲述

左程云

链表入门题目-划分链表

前置知识：理解链表及其基本调整　　建议：做过这个题的同学跳过

给你一个链表的头节点 `head` 和一个特定值 `x`
请你对链表进行分隔，使得所有 小于 `x` 的节点都出现在 大于或等于 `x` 的节点之前
你应当 保留 两个分区中每个节点的初始相对位置

链表题目在笔试、面试中的意义就是检验coding能力如何
更难的题目会在【必备】课程里讲述

左程云

队列和栈-链表、数组实现

前置知识：链表
建议：比较初级，会的可以跳过，但是要注意环形队列用数组实现这个高频考点

1）队列的介绍
2）栈的介绍
3）队列的链表实现和数组实现
4）栈的数组实现
5）环形队列用数组实现

队列、栈、双端队列可以组成非常多重要的数据结构
将在【必备】课程里继续

左程云

队列和栈入门题目-栈和队列相互实现

用栈实现队列

请你仅使用两个栈实现先入先出队列。队列应当支持一般队列支持的所有操作（push、pop、peek、empty）：

实现 MyQueue 类：

- void push(int x) 将元素 x 推到队列的末尾
- int pop() 从队列的开头移除并返回元素
- int peek() 返回队列开头的元素
- boolean empty() 如果队列为空，返回 true ；否则，返回 false

说明：

- 你 只能 使用标准的栈操作 -- 也就是只有 push to top, peek/pop from top, size, 和 is empty 操作是合法的
- 你所使用的语言也许不支持栈。你可以使用 list 或者 deque（双端队列）来模拟一个栈，只要是标准的栈操作即可

队列和栈入门题目-栈和队列相互实现

用队列实现栈

请你仅使用两个队列实现一个后入先出（LIFO）的栈，并支持普通栈的全部四种操作（push、top、pop 和 empty）。

实现 MyStack 类：

- void push(int x) 将元素 x 压入栈顶。
- int pop() 移除并返回栈顶元素。
- int top() 返回栈顶元素。
- boolean empty() 如果栈是空的，返回 true ；否则，返回 false 。

注意：

- 你只能使用队列的基本操作 —— 也就是 push to back、peek/pop from front、size 和 is empty 这些操作
- 你所使用的语言也许不支持队列。 你可以使用 list （列表）或者 deque（双端队列）来模拟一个队列 , 只要是标准的队列操作即可

栈的入门题目-最小栈

最小栈

设计一个支持 push ，pop ，top 操作，并能在常数时间内检索到最小元素的栈。

实现 MinStack 类:

- MinStack() 初始化栈对象。
- void push(int val) 将元素val推入堆栈。
- void pop() 删除栈顶部的元素。
- int top() 获取栈顶部的元素。
- int getMin() 获取栈中的最小元素。

双端队列-双链表和固定数组实现

前置知识：双链表
建议：用双链表实现双端队列比较初级，会的可以跳过；但是用固定数组的实现双端队列值得听

1）双端队列的介绍
2）双端队列用双链表的实现
3）双端队列用固定数组的实现

队列、栈、双端队列可以组成非常多重要的数据结构
将在【必备】课程里继续

左程云

二叉树及其三种序的递归实现

前置知识：无
建议：会同学可以跳过

1）二叉树的节点
2）二叉树的先序、中序、后序
3）递归序加工出三种序的遍历
4）时间复杂度0(n)，额外空间复杂度0(h)，h是二叉树的高度

关于递归更多的内容会在【必备】课程里继续
二叉树更多更难的题会在【必备】课程里继续

左程云

二叉树遍历的非递归实现和复杂度分析

前置知识：二叉树、先序、中序、后序、栈
建议：不要跳过

1）用栈实现二叉树先序遍历
2）用栈实现二叉树中序遍历
3）用两个栈实现二叉树后序遍历，好写但是不推荐，因为需要收集所有节点，最后逆序弹出，额外空间复杂度为O(n)
4）用一个栈实现二叉树后序遍历
5）遍历二叉树复杂度分析：
    a. 时间复杂度O(n)，递归和非递归都是每个节点遇到有限几次，当然O(n)
    b. 额外空间复杂度O(h)，递归和非递归都需要二叉树高度h的空间来保存路径，方便回到上级去
    c. 存在时间复杂度O(n)，额外空间复杂度O(1)的遍历方式：Morris遍历
    d. Morris遍历比较难，也比较冷门，会在【扩展】课程里讲述

关于递归更多的内容会在【必备】课程里继续
二叉树更多更难的题会在【必备】课程里继续

左程云

算法笔试中处理输入和输出

前置知识：无

1）填函数风格
2）acm风格（笔试、比赛最常见）
   a. 规定数据量(BufferedReader、StreamTokenizer、PrintWriter)，其他语言有对等的写法
   b. 按行读(BufferedReader、PrintWriter)，其他语言有对等的写法
   c. 不要用Scanner、System.out，IO效率慢
3）不推荐：临时动态空间
4）推荐：全局静态空间

左程云

递归和master公式

前置知识：无

1）从思想上理解递归：对于新手来说，递归去画调用图是非常重要的，有利于分析递归
2）从实际上理解递归：递归不是玄学，底层是利用系统栈来实现的
3）任何递归函数都一定可以改成非递归，不用系统帮你压栈（系统栈空间），自己压栈呗（内存空间）
4）递归改成非递归的必要性：
    a. 工程上几乎一定要改，除非确定数据量再大递归也一定不深，归并排序、快速排序、线段树、很多的平衡树等，后面都讲
    b. 算法笔试或者比赛中（能通过就不改）
5）master公式
    a. 所有子问题规模相同的递归才能用master公式，T(n) = a * T(n/b) + O(n^c)，a、b、c都是常数
    b. 如果log(b,a) < c，复杂度为：O(n^c)
    c. 如果log(b,a) > c，复杂度为：O(n^log(b,a))
    d. 如果log(b,a) == c，复杂度为：O(n^c * logn)
6）一个补充
    T(n) = 2*T(n/2) + O(n*logn)，时间复杂度是O(n * ((logn)的平方))，证明过程比较复杂，记住即可

左程云

归并排序

前置知识：讲解019-算法笔试中处理输入和输出、讲解020-递归和master公式

1）左部分排好序、右部分排好序、利用merge过程让左右整体有序
2）merge过程：谁小拷贝谁，直到左右两部分所有的数字耗尽，拷贝回原数组
3）递归实现和非递归实现
4）时间复杂度O(n * logn)
5）需要辅助数组，所以额外空间复杂度O(n)
6）归并排序为什么比O(n^2)的排序快？因为比较行为没有浪费！
7）利用归并排序的便利性可以解决很多问题 - 归并分治 - 下节课
注意：
有些资料说可以用原地归并排序，把额外空间复杂度变成O(1)，不要浪费时间去学
因为原地归并排序确实可以省空间，但是会让复杂度变成O(n^2)
有关排序更多的概念、注意点、闭坑指南，将在后续课程继续

左程云

归并分治

前置知识：讲解021-归并排序

原理：
1）思考一个问题在大范围上的答案，是否等于，左部分的答案 + 右部分的答案 + 跨越左右产生的答案
2）计算“跨越左右产生的答案”时，如果加上左、右各自有序这个设定，会不会获得计算的便利性
3）如果以上两点都成立，那么该问题很可能被归并分治解决（话不说满，因为总有很毒的出题人）
4）求解答案的过程中只需要加入归并排序的过程即可，因为要让左、右各自有序，来获得计算的便利性

补充：
1）一些用归并分治解决的问题，往往也可以用线段树、树状数组等解法。时间复杂度也都是最优解，这些数据结构都会在【必备】或者【扩展】课程阶段讲到
2）本节讲述的题目都是归并分治的常规题，难度不大。归并分治不仅可以解决简单问题，还可以解决很多较难的问题，只要符合上面说的特征。比如二维空间里任何两点间的最短距离问题，这个内容会在【挺难】课程阶段里讲述。顶级公司考这个问题的也很少，因为很难，但是这个问题本身并不冷门，来自《算法导论》原题
3）还有一个常考的算法：“整块分治”。会在【必备】课程阶段讲到

聊：精妙又美丽的思想传统

左程云

归并分治-小和问题

假设数组 s = [1, 3, 5, 2, 4, 6]
在s[0]的左边所有 <= s[0]的数的总和为0
在s[1]的左边所有 <= s[1]的数的总和为1
在s[2]的左边所有 <= s[2]的数的总和为4
在s[3]的左边所有 <= s[3]的数的总和为1
在s[4]的左边所有 <= s[4]的数的总和为6
在s[5]的左边所有 <= s[5]的数的总和为15
所以s数组的“小和”为 : 0 + 1 + 4 + 1 + 6 + 15 = 27
给定一个数组arr，实现函数返回arr的“小和”
测试链接 :
https://www.nowcoder.com/practice/edfe05a1d45c4ea89101d936cac32469

课上解法和归并排序的时间复杂度一样O(n * logn)

归并分治-翻转对数量

给定一个数组 nums ，

如果 i < j 且 nums[i] > 2*nums[j] 我们就将 (i, j) 称作一个翻转对

你需要返回给定数组中的翻转对的数量

测试链接 ：

https://leetcode.cn/problems/reverse-pairs/

课上解法和归并排序的时间复杂度一样O(n * logn)

左程云

随机快速排序

前置知识：讲解007-时间复杂度和空间复杂度-分析随机行为的时间复杂度的部分

经典随机快速排序流程讲解

荷兰国旗问题优化随机快速排序流程讲解

荷兰国旗问题优化后的过程：
在当前范围上选择一个数字x，利用荷兰国旗问题进行数组的划分，<x =x >x
对<x范围重复这个过程，对>x范围重复这个过程

荷兰国旗问题的优化点：选出一个数字x，数组在划分时会搞定所有值是x的数字

左程云

随机快速排序

快速排序的时间和空间复杂度分析

核心点：怎么选择数字？

选择的数字是当前范围上的固定位置，比如范围上的最右数字，那么就是普通快速排序
选择的数字是当前范围上的随机位置，那么就是随机快速排序

普通快速排序，时间复杂度O(n^2)，额外空间复杂度O(n)
随机快速排序，时间复杂度O(n * logn)，额外空间复杂度O(logn)

关于复杂度的分析，进行定性的说明，定量证明略，因为证明较为复杂

算法导论-7.4.2有详细证明

随机选择算法

前置知识：讲解023-随机快速排序

无序数组中寻找第K大的数
给定整数数组 nums 和整数 k，请返回数组中第 k 个最大的元素。
请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。
你必须设计并实现时间复杂度为 O(n) 的算法解决此问题。

利用改写快排的方法，时间复杂度O(n)，额外空间复杂度O(1)
上面问题的解法就是随机选择算法，是常考内容！本视频定性讲述，定量证明略，算法导论-9.2有详细证明
不要慌！
随机快速排序、随机选择算法，时间复杂度的证明理解起来很困难，只需记住结论，但并不会对后续的算法学习造成什么影响
因为数学很好才能理解的算法和数据结构其实比较少，绝大部分的内容都只需要高中数学的基础就能理解

算法导论第9章，还有一个BFPRT算法，不用随机选择一个数的方式，也能做到时间复杂度O(n)，额外空间复杂度O(log n)
早些年我还讲这个算法，不过真的很冷门，很少在笔试、面试、比赛场合出现，所以算了。有兴趣的同学可以研究一下

堆结构和堆排序

前置知识：无

堆结构
完全二叉树和数组前缀范围来对应，大小，单独的变量size来控制
i的父亲节点：(i-1)/2，i的左孩子：i*2 + 1，i的右孩子：i*2 + 2
堆的定义（大根堆、小根堆），本节课讲解按照大根堆来讲解，小根堆是同理的。
堆的调整：heapInsert（向上调整）、heapify（向下调整）
heapInsert、heapify方法的单次调用，时间复杂度O(log n)，完全二叉树的结构决定的

堆排序
A. 从顶到底建堆，时间复杂度O(n * log n)，log1 + log2 + log3 + … + logn -> O(n*logn)
或者用增倍分析法：建堆的复杂度分析+子矩阵数量的复杂度分析
B. 从底到顶建堆，时间复杂度O(n)，总代价就是简单的等比数列关系，为啥会有差异？简单图解一下
C. 建好堆之后的调整阶段，从最大值到最小值依次归位，时间复杂度O(n * log n)
时间复杂度O(n * log n)，不管以什么方式建堆，调整阶段的时间复杂度都是这个，所以整体复杂度也是这个
额外空间复杂度是O(1)，因为堆直接建立在了要排序的数组上，所以没有什么额外空间

注意：堆结构比堆排序有用的多，尤其是和比较器结合之后。后面几节课会重点讲述。

哈希表、有序表和比较器的用法

前置知识：无
提醒：讲解虽然用的java语言，但是任何语言都有对等的概念
提醒：后续有专门的章节来详解哈希函数、有序表，这节课就是常规用法展示

哈希表的用法（认为是集合，根据值来做key 或者 根据内存地址做key）
HashSet和HashMap原理一样，有无伴随数据的区别
增、删、改、查时间为O(1)，但是大常数
所以当key的范围是固定的、可控的情况下，可以用数组结构替代哈希表结构
注意：
Java中通过自定义hashCode、equals等方法
任何类都可以实现“根据值做key”或者“根据内存地址做key”的需求
但是这里不再展开，因为在算法学习这个范畴内，这些并不重要，还有其他语言的同学也不关心这些
笔试、面试、比赛也都不会用到，课上只说对算法学习重要的内容

左程云

哈希表、有序表和比较器的用法

前置知识：无
提醒：讲解虽然用的java语言，但是任何语言都有对等的概念
提醒：后续有专门的章节来详解哈希函数、有序表，这节课就是常规用法展示

有序表的用法（认为是集合，但是有序组织）
TreeSet和TreeMap原理一样，有无伴随数据的区别
增、删、改、查 + 很多和有序相关的操作(floor、ceilling等)，时间为O(log n)
有序表比较相同的东西会去重，如果不想去重就加入更多的比较策略（比较器定制）。堆不会去重。
有序表在java里就是红黑树实现的
AVL树、SB树、替罪羊树、Treap、Splay、跳表等等很多结构都可实现同样功能
后续的课程会涉及，这里不做展开，只讲解简单用法

比较器：定制比较策略。用在排序、堆、有序表等很多需要序的结构中都可使用
定义类、直接Lamda表达式
字典序的概念

左程云

堆结构常见题

前置知识：讲解025-堆结构和堆排序、讲解026-比较器

题目1
合并K个有序链表

题目2
线段最多重合问题

题目3
让数组整体累加和减半的最少操作次数

左程云

基数排序

前置知识：无

基于比较的排序
只需要定义好两个对象之间怎么比较即可，对象的数据特征并不关心，很通用
不基于比较的排序
和比较无关的排序，对于对象的数据特征有要求，并不通用

计数排序，非常简单，但是数值范围比较大了就不行了

基数排序的实现细节，非常优雅的一个实现
关键点：前缀数量分区的技巧、数字提取某一位的技巧
时间复杂度O(n)，额外空间复杂度O(m)，需要辅助空间做类似桶的作用，来不停的装入、弹出数字

一般来讲，计数排序要求，样本是整数，且范围比较窄
一般来讲，基数排序要求，样本是10进制的非负整数
如果不是就需要转化，代码里做了转化，并且代码里可以设置任何进制来进行排序
一旦比较的对象不再是常规数字，那么改写代价的增加是显而易见的，所以不基于比较的排序并不通用

左程云

重要排序算法的总结

前置知识：之前讲的所有排序，本节课涉及的所有排序，之前的视频都讲了

稳定性
排序算法的稳定性是指：同样大小的样本在排序之后不会改变原始的相对次序
每个算法都说明一下
稳定性对基础类型对象来说毫无意义；稳定性对非基础类型对象有意义，可以保留之前的相对次序

主要算法时间、空间、稳定性总结

	时间	空间	稳定性
SelectionSort	O(N^2)	O(1)	无
BubbleSort	O(N^2)	O(1)	有
InsertionSort	O(N^2)	O(1)	有
MergeSort	O(N*logN)	O(N)	有
QuickSort	O(N*logN)	O(logN)	无
HeapSort	O(N*logN)	O(1)	无
CountSort	O(N)	O(M)	有
RadixSort	O(N)	O(M)	有

注意：随机快速排序的复杂度一定要按照概率上的期望指标来估计，用最差的复杂度估计无意义，随机快排讲解视频里已经有详细的说明

左程云

重要排序算法的总结

基于比较的排序，时间复杂度O(n*logn)，空间复杂度低于O(n)，还具有稳定性的排序算法目前没有找到
TimSort也不行，虽然在实际应用中TimSort通常不需要这么多的额外空间，但空间复杂度指标就是O(n)
有兴趣的同学可以研究，但是在算法面试、笔试、比赛中都很少用到TimSort算法
同时还有希尔排序(ShellSort)也不常用，有兴趣的同学可以研究一下，就是加入步长调整的插入排序

所以，一切看你在排序过程中在乎什么

数据量非常小的情况下可以做到非常迅速：插入排序
性能优异、实现简单且利于改进（面对不同业务可以选择不同划分策略）、不在乎稳定性：随机快排
性能优异、不在乎额外空间占用、具有稳定性：归并排序
性能优异、额外空间占用要求O(1)、不在乎稳定性：堆排序

异或运算的骚操作

前置知识：讲解003-二进制和位运算
特别提醒：Python的同学实现位运算的题目需要特别注意，需要自己去手动处理溢出和符号扩展等问题
比如：(n << shift_amount) & 0xFFFFFFFF

先来一个好玩的问题：
袋子里一共a个白球，b个黑球，每次从袋子里拿2个球，每个球每次被拿出机会均等
如果拿出的是2个白球、或者2个黑球，那么就往袋子里重新放入1个白球
如果拿出的是1个白球和1个黑球，那么就往袋子里重新放入1个黑球
那么最终袋子里一定会只剩1个球，请问最终的球是黑的概率是多少？用a和b来表达这个概率。

被镇住了吧？其实这题是一个陷阱。

答案：
黑球的数量如果是偶数，最终的球是黑的概率是0%
黑球的数量如果是奇数，最终的球是黑的概率是100%
完全和白球的数量无关。为啥？异或运算的性质了解之后，就了解了。

异或运算的骚操作

异或运算性质

1）异或运算就是无进位相加
2）异或运算满足交换律、结合律，也就是同一批数字，不管异或顺序是什么，最终的结果都是一个
3）0^n=n，n^n=0
4）整体异或和如果是x，整体中某个部分的异或和如果是y，那么剩下部分的异或和是x^y

这些结论最重要的就是1）结论，所有其他结论都可以由这个结论推论得到

其中第4）相关的题目最多，利用区间上异或和的性质

Nim博弈也是和异或运算相关的算法，将在后续【必备】课程里讲到

异或运算的骚操作

题目1 交换两个数
题目2 不用任何判断语句和比较操作，返回两个数的最大值
题目3 找到缺失的数字
题目4 数组中1种数出现了奇数次，其他的数都出现了偶数次，返回出现了奇数次的数

Brian Kernighan算法 - 提取出二进制状态中最右侧的1
题目5 数组中有2种数出现了奇数次，其他的数都出现了偶数次，返回这2种出现了奇数次的数

题目6 数组中只有1种数出现次数少于m次，其他数都出现了m次，返回出现次数小于m次的那种数

左程云

位运算的骚操作

前置知识：讲解003-二进制和位运算、讲解030-异或运算的骚操作
特别提醒：Python的同学实现位运算的题目需要特别注意，需要自己去手动处理溢出和符号扩展等问题
比如：`(n << shift_amount) & 0xFFFFFFFF`

位运算有很多奇技淫巧，位运算的速度非常快，仅次于赋值操作，常数时间极好！

这节课展示一下先贤的功力！骚就完了！

关于位运算还有非常重要的内容：
N皇后问题用位运算实现，将在【必备】课程里讲到
状态压缩的动态规划，将在【扩展】课程里讲到

位运算的骚操作

题目1 判断一个整数是不是2的幂
题目2 判断一个整数是不是3的幂
题目3 返回大于等于n的最小的2的幂
题目4 区间[left, right]内所有数字 & 的结果

题目5 反转一个二进制的状态，不是0变1、1变0，是逆序。超自然版
题目6 返回一个数二进制中有几个1。超自然版，看完佩服大牛的脑洞，能爽一整天

题目5和题目6代码看着跟脑子有大病一样，承认很强但似乎有点太嗝瑟了，是这样吗？
不是的，条件判断相比于赋值、位运算、算术运算是稍慢的，所以其实有现实意义
但是不需要追求在练算法过程中尽量少写条件判断，
那样会带来很多不必要的困扰，还是要写尽量直白、尤其是自己能理解的代码最好
大牛的实现欣赏完理解就好，下次当模版直接用

左程云

位图

前置知识：讲解003-二进制和位运算、讲解005-对数器
特别提醒：Python的同学实现位运算的题目需要特别注意，需要自己去手动处理溢出和符号扩展等问题
比如：(n << shift_amount) & 0xFFFFFFFF

位图原理
其实就是用bit组成的数组来存放值，用bit状态1、0代表存在、不存在，取值和存值操作都用位运算
限制是必须为连续范围且不能过大。好处是极大的节省空间，因为1个数字只占用1个bit的空间。

位图的实现
Bitset(int n)：初始化位图的大小，只支持0~n-1所有数字的增删改查
void add(int num)：把num加入到位图
void remove(int num)：把num从位图中删除
void reverse(int num)：如果位图里没有num，就加入；如果位图里有num，就删除
boolean contains(int num)：查询num是否在位图中
将采用对数器验证，当你找不到测试链接的时候就用对数器验证，而且对数器验证更稳妥、更能练习debug能力

左程云

位图

找到了一个相关测试：https://leetcode-cn.com/problems/design-bitset/

Bitset是一种能以紧凑形式存储位的数据结构
Bitset(int n) : 初始化n个位，所有位都是0
void fix(int i) : 将下标i的位上的值更新为1
void unfix(int i) : 将下标i的位上的值更新为0
void flip() : 翻转所有位的值
boolean all() : 是否所有位都是1
boolean one() : 是否至少有一位是1
int count() : 返回所有位中1的数量
String toString() : 返回所有位的状态

位图的后续扩展，将在【扩展】课程里进一步讲述

位运算实现加减乘除

前置知识：讲解003-二进制和位运算
特别提醒：Python的同学实现位运算的题目需要特别注意，需要自己去手动处理溢出和符号扩展等问题
比如：(n << shift_amount) & 0xFFFFFFFF

位运算实现加减乘除，过程中不能出现任何算术运算符

加法：利用每一步无进位相加的结果 + 进位的结果不停计算，直到进位消失
减法：利用加法，和一个数字x相反数就是(~x)+1
乘法：回想小学时候怎么学的乘法，除此之外没别的了
除法：为了防止溢出，让被除数右移，而不是除数左移。从高位尝试到低位。

链表高频题目和必备技巧

前置知识：
讲解009~012-链表入门内容、讲解021-归并排序
讲解026-哈希表的使用、讲解029-排序算法的稳定性

链表类题目注意点：
1，如果笔试中空间要求不严格，直接使用容器来解决链表问题
2，如果笔试中空间要求严格、或者在面试中面试官强调空间的优化，需要使用额外空间复杂度O(1)的方法
3，最常用的技巧-快慢指针
4，链表类题目往往都是很简单的算法问题，核心考察点也并不是算法设计，是coding能力
5，这一类问题除了多写多练没有别的应对方法

个人建议：链表类问题既然练的就是coding，那么不要采取空间上讨巧的方式来练习

注意：链表相关的比较难的问题是约瑟夫环问题，会在【扩展】阶段讲解，变形很多会单独出一期视频讲解

左程云

链表高频题目和必备技巧

题目1 ： 返回两个无环链表相交的第一个节点

题目2 ： 每k个节点一组翻转链表

题目3 ： 复制带随机指针的链表

题目4 ： 判断链表是否是回文结构。这个题的流程设计甚至是考研常用。快慢指针找中点。

题目5 ： 返回链表的第一个入环节点。快慢指针找中点。

题目6 ： 在链表上排序。要求时间复杂度O(n * log n)，额外空间复杂度O(1)，还要求排序有稳定性。

注意：
这些题目往往难度标为“简单”，是因为用容器解决真的很简单
但是不用容器、实现额外空间复杂度O(1)的方法并不轻松，包括很多提交的答案也都没有符合要求

数据结构设计高频题

前置知识：
讲解007-动态数组和扩容分析、讲解009~012-链表入门内容
讲解025-堆结构、讲解026-哈希表、有序表、比较器的使用

注意：
本节以数据结构设计高频题为主，并不涉及太难的数据结构设计题目
数据结构设计的更难题目，需要学习更多数据结构之后才能解决，如前缀树、并查集、线段树等
后续会更新【必备】、【扩展】、【挺难】视频，包含大量基础和高级数据结构最好懂的原理详解
更多更难的算法、数据结构题目会在“好题解析”系列进行讲解，这个系列将在原理详解系列结束后开始

欢迎持续关注、转发

左程云

数据结构设计高频题

题目1 ： setAll功能的哈希表

题目2 ： 实现LRU结构

题目3 ： 插入、删除和获取随机元素O(1)时间的结构

题目4 ： 插入、删除和获取随机元素O(1)时间且允许有重复数字的结构

题目5 ： 快速获得数据流的中位数的结构

题目6 ： 最大频率栈

题目7 ： 全O(1)的数据结构

左程云

二叉树高频题目-上-不含树型dp

前置知识：
讲解013-队列用数组实现、讲解017~018-二叉树入门内容

特别说明：
这一期和下一期视频，会讲解二叉树高频题目，但是不含树型dp的题目
树型dp问题，会放在【必备】课程的动态规划大章节部分讲述
树型dp中的换根dp问题，会放在【扩展】课程的动态规划大章节部分讲述
AVL树的实现，树的左旋、右旋，这些内容也会在【扩展】课程里讲述

二叉树高频题目-上-不含树型dp

题目1 ： 二叉树的层序遍历
题目2 ： 二叉树的锯齿形层序遍历
题目3 ： 二叉树的最大特殊宽度
题目4 ： 求二叉树的最大深度、求二叉树的最小深度
题目5 ： 二叉树先序序列化和反序列化
题目6 ： 二叉树按层序列化和反序列化
题目7 ： 利用先序与中序遍历序列构造二叉树
题目8 ： 验证完全二叉树
题目9 ： 求完全二叉树的节点个数，要求时间复杂度低于O(n)

注意：中序遍历无法完成二叉树的序列化和反序列化，代码中给出了说明。后序遍历可以但不再详述。

左程云

二叉树高频题目-下-不含树型dp

前置知识：
讲解013-队列用数组实现、讲解017~018-二叉树入门内容

特别说明：
这一期和上一期视频，会讲解二叉树高频题目，但是不含树型dp的题目
树型dp问题，会放在【必备】课程的动态规划大章节部分讲述
树型dp中的换根dp问题，会放在【扩展】课程的动态规划大章节部分讲述
AVL树的实现，树的左旋、右旋，这些内容也会在【扩展】课程里讲述

左程云

二叉树高频题目-下-不含树型dp

题目1 ： 普通二叉树上寻找两个节点的最近公共祖先
题目2 ： 搜索二叉树上寻找两个节点的最近公共祖先
题目3 ： 收集累加和等于aim的所有路径(递归恢复现场)
题目4 ： 验证平衡二叉树(树型dp沾边)
题目5 ： 验证搜索二叉树(树型dp沾边)
题目6 ： 修剪搜索二叉树
题目7 ： 二叉树打家劫舍问题(树型dp沾边)

注意 ： 问题1又叫lca问题，非常重要的问题！
Tarjan算法解决lca的批量查询、树链剖分算法解决lca的在线查询，会在【扩展】课程讲述
注意 ： 数组的打家劫舍问题变形很多，会在【必备】课程的动态规划大章节部分讲述
注意 ： 再次强调树型dp的整体讲解，会在【必备】课程的动态规划大章节部分讲述

常见经典递归过程解析

前置知识：
讲解017、020、021、023、036、037
这些章节都分析过递归，不熟悉的同学可以先熟悉一下

带路径的递归 vs 不带路径的递归(大部分dp，状态压缩dp认为是路径简化了结构，dp专题后续讲述)
任何递归都是dfs且非常灵活。回溯这个术语并不重要。

题目1 ： 返回字符串全部子序列，子序列要求去重。时间复杂度O(2^n * n)
题目2 ： 返回数组的所有组合，可以无视元素顺序。时间复杂度O(2^n * n)
题目3 ： 返回没有重复值数组的全部排列。时间复杂度O(n! * n)
题目4 ： 返回可能有重复值数组的全部排列，排列要求去重。时间复杂度O(n! * n)
题目5 ： 用递归逆序一个栈。时间复杂度O(n^2)
题目6 ： 用递归排序一个栈。时间复杂度O(n^2)
题目7 ： 打印n层汉诺塔问题的最优移动轨迹。时间复杂度O(2^n)

左程云

嵌套类问题的递归解题套路

前置知识：
讲解017、020、021、023、036、037、038
这些章节都分析过递归，尤其讲解038，不熟悉的同学可以先熟悉一下

嵌套类问题的解题套路
大概过程：
1) 定义全局变量 int where
2) 递归函数f(i) : s[i..]，从i位置出发开始解析，遇到 字符串终止 或 嵌套条件终止 就返回
3) 返回值是f(i)负责这一段的结果
4) f(i)在返回前更新全局变量where，让上级函数通过where知道解析到了什么位置，进而继续

执行细节：
1) 如果f(i)遇到 嵌套条件开始，就调用下级递归去处理嵌套，下级会负责嵌套部分的计算结果
2) f(i)下级处理完成后，f(i)可以根据下级更新的全局变量where，知道该从什么位置继续解析

左程云

嵌套类问题的递归解题套路

实战一下

题目1 ： 含有嵌套的表达式求值。时间复杂度0(n)

题目2 ： 含有嵌套的字符串解码。时间复杂度0(n)

题目3 ： 含有嵌套的分子式求原子数量。时间复杂度0(n)

左程云

N皇后问题-重点是位运算的版本

前置知识
递归相关 ： 讲解038
位运算相关 ： 讲解003、030、031、032、033

解决N皇后问题的时间复杂度是O(n!)，好的方法可以大量剪枝，大量优化常数时间

用数组表示路径的方法（经典、常数时间慢，不推荐）
1) 记录之前每一行的皇后放在了什么列
2) 来到第i行的时候，可以根据0..i-1行皇后的位置，判断能放哪些列
3) 把能放的列都尝试一遍，每次尝试修改路径数组表示当前的决策，后续返回的答案都累加返回

用位运算的方法（巧妙、常数时间快，推荐）
1) int col : 0..i-1行皇后放置的位置因为正下方向延伸的原因，哪些列不能再放皇后
2) int left : 0..i-1行皇后放置的位置因为左下方向延伸的原因，哪些列不能再放皇后
3) int right : 0..i-1行皇后放置的位置因为右下方向延伸的原因，哪些列不能再放皇后
4) 根据col、left、right，用位运算快速判断能放哪些列
5) 把能放的列都尝试一遍，每次尝试修改3个数字表示当前的决策，后续返回的答案都累加返回

左程云