面试算法

# 第一讲：算法面试

## 第一节 何为算法

数据结构分为两个层面：思维逻辑结构层面和具体实现层面

比如快速排序算法，思维逻辑层面与二叉排序树是一致的，但具体实现的时候并没有用到二叉排序树

算法也分为两个层面：思维逻辑结构层面和具体实现层面

人需要先会这些算法，然后才能教计算机去做算法

## 第二节 何为面试

面试是选拔人才的一种手段，笔试是另一种手段。因此当面试的内容与笔试的内容差不多时，这样的面试是一场失败的面试，若面试的内容与笔试的内容互补时，面试算是成功。

简历容易出现**“概念不对齐”**，比如精通C++，每个人对“精通”的理解不同

笔试也容易出现“概念不对齐”，需要用面试来调整

面试是个**不平等对抗**的场景。面试官出现知识盲点，他会一句带过，可面试者出现知识盲点时，面试就game over了。

好的面试官是允许面试者出现知识盲点，但不允许在面试官的引导下在面试者已有的知识范围内，面试者仍然得不到结果（允许不会，但不允许在启发下还不会）

## 第三节 算法思维

何为算法思维？

首先，算法思维是一种感觉

一个算法问题不是说你不会做，而是你不知道怎么告诉计算机去做。（如果你本身就不会做，那就更不用说教计算机了）

Hash函数：

引入数组，数组是按索引来获取数值（属性：index和val），时间复杂度为O(1)，若将字符串转换成数组存储，将能快速寻找答案。

问题：abc –> 映射关系 –> index

“abc”在计算机中的表示是01表示，因此问题就转变成**如何用一个数值表示多个数值？（高维空间映射到低维空间， 离散化）**

高维空间映射到低维空间的出现的问题：**一个位置会出现多个数值（10鸡蛋放到9个篮子里面）**

Hash表需要解决的两个问题：1.如何进行映射 2.如何解决冲突

“永不停机定理”，图灵机停机问题

停机问题：能否为计算机X编程，使其在有限的时间内判断出计算Y（装有另一个程序以及一些初始数据）是否会停止运行？

**火柴棍游戏（数据表示 + 移动方法）**

**数字6008，如何移动两根火柴棍，使得它表示的数字尽可能的大？**

题目的本质：一个数怎样表示才是尽可能的大？（讨论数字表示的问题）

思路：

0. 使前面的位数尽可能的大

1. 增加1位: 61005

2. 使用科学计数法： 6E88（6\*10^88）

3. 使用两次科学计数法： 9ee8（9\*10^(10^8)）

暗示一个道理：只有你人能想到各种各种的方法来解决这个问题，才能告诉计算机怎样会解决这个问题，如果你人会解决这个问题采用的方法不是最优，那么计算机程序无论如何也不可能是最优的。

（或者是，你可以不知道正确答案，但你必须知道答案的范围，才能告诉计算机如何在这个范围里面搜索正确的答案）

**成本与收益（算法研究的重点）**

具体问题具体分析

衡量时间和空间的代价

**算法思维是长期锻炼而来的一种感觉**

算法是什么？

算法为什么这样做？

算法怎么来的？

## 第四节 算法优化

### 1. **绝对值与相对值的转换**

例子1：考虑一个积性函数F(a,b)，求从a到b的连乘或者连加

转化： F(a,b) = F(1,b) – F(1,a-1)

假设a有n的取值，b有n的取值，若存储F(a,b)的值，需要存储n\*n个值

而存储F(1,b)和F(1,a-1)只需存储n个值。最后将n^2的问题转换成n的问题

例子2：字典树

字典树是哈希表的特例

数据的表示形式到底合不合理？

用01来表示万物

### 2. 记录式与计算式的转换

生活的例子：1+1=2,1+2=3, 那么 1+3=？

记录式的方式：把1+1=2,1+2=3都存储下来，但不管他背后的运算规则，所以1+3的结果自然无法计算

计算式的方式：把算式背后的原理和运算规则记录下来，那么1+3根据规则便可直接求算。

字典树 ——> 双数组字典树：将边的信息用计算的方式求出，而不是直接记录

例子：自然语言处理当中的近义词处理

从记录式，直接记录两个词的关系，到计算式，将两个词转换成向量，再求两个向量的相似度。

### 3. 寻找等价问题

问题：有一个长串，有一个短串，问短串是否是长串的子串？

问题2：在不知道KMP算法的情况下，如何推导出KMP算法

思路1：直接循环判断短串是否在长串出现







## 第五节 吐槽内容

# 第二讲：线性表

算法口诀：难题首选**动归**，受阻**贪心暴力**；考虑**分治**思想，配合**排序哈希**

**1）动态规划**是解决相当数目问题的法宝；

a) **滚动数组**降低空间复杂度

**2）贪心法**并不简单

a) Dijkstra最短路径、最小生成数Prim、Kruskal算法

**3）深度优先搜索、广度优先搜索**，都可以归结为**暴力求解**；

a) **分支限界**条件加快搜索效率

**4）分治法**在降低问题规模问题上很有效；

a) 快速排序、归并排序——**递归、广义分治法**

**5）排序**是为了更好的查找；

a) 各种排序方法的选择

**6）**实在不行了，**空间换时间——Hash**

**a) 深入理解Hash——int a[65536] / int a[256]**

**7）**有些题目需要上述两者或者多个技术**综合运用**。

## 链表

1. 链表相加

2. 链表部分翻转

3. 链表去重

4. 链表划分

5. 链表公共结点

6. 从尾到头打印链表

**一、 链表相加**

题目1：给定两个链表，分别表示两个非负整数。它们的数字逆序存储在链表中，且每个结点只存储一个数字，计算两个数的和，并且返回和的链表头指针。

如：输入：2🡪4🡪3(即342)、5🡪6🡪4（即465）输出：7🡪0🡪8（即807）

思路：

1）处理按位相加，超出10的部分，需要进位

2）处理长串，直接将长串剩余的部分添加到链表的后面

3）处理进位，当进位不为0时，需要新增一位

进一步分析与思考：

1. 利用这个结构实现大整数运算？

思路：

1）将两个大整数逆序存储到链表当中

2）按位相加处理

2. 如何实现乘法？

**二、 链表部分翻转**

题目2：给定一个链表，翻转该链表从m到n的位置。要求直接翻转而非申请新空间。

如：给定1🡪2🡪3🡪4🡪5， m=2，n=4，返回1🡪4🡪3🡪2🡪5.

假定给出的参数满足：1<=m<=n<=链表长度。

思路：

找到待翻转的前一个位置，即1和2之间，然后把3和4，一个一个地提前的2的前面去。（空间为1）

**三、 链表去重**

题目3：给定排序的链表，删除重复元素，只保留重复元素第一次出现的结点。

如：给定：2🡪3🡪3🡪5🡪7🡪8🡪8🡪8🡪9🡪9🡪10

返回：2🡪3🡪5🡪7🡪8🡪9🡪10

思路：如果没有排序，需要先排序

题目4：给定排序的链表，删除全部的重复元素，包括元素本身

如：给定：2🡪3🡪3🡪5🡪7🡪8🡪8🡪8🡪9🡪9🡪10

返回：2🡪5🡪7🡪10

思路：从右往左删除

**四、 链表划分**

题目：给定一个链表和一个值x，将链表划分成两个部分，使得划分后小于x的结点在前，大于等于x的结点在后。在这两部分中要保持原链表中的出现顺序。

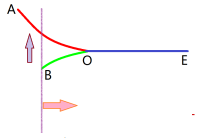
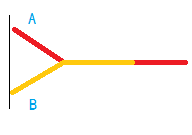
如：给定链表1🡪4🡪3🡪2🡪5🡪2和x=3，返回1🡪2🡪2🡪4🡪3🡪5

思路：分别申请两个指针p1和p2，小于x的添加到p1中，大于等于x的添加到p2中；最后，将p2链接到p1的末端即可。

拓展：单链表的排序，可以用**快速排序**来完成

**五、 链表公共结点**

题目：给定两个单向链表，计算两个链表的第一个公共结点，若没有公共节点，返回空

思路：

**（两个链表从第一个公共结点到链表的尾结点是完全重合的）**

第一种情况：长的单链表为m，短的单链表为n，先让长的移动|m-n|位，然后让两者齐头并进，直至找到第一个公共节点

第二种情况：从两者的最后一位出发，若相等，则两者的指针同时往前移位，若不相等，则向前移动一位长的指针。（不会出现）

相关应用：

求树中任意两个结点的最近公共祖先

**小结：**

1） 单链公共结点问题中，如果是链表存在环，则需要使用**快慢指针**的方式计算公共结点。

即两个指针，每次分布移动一个/两个结点

2） 可以发现，纯链表的题目，往往不难，但需要扎实的Coding基本功，在实现过程中，要特别小心next的指向，此外，删除结点时，一定要确保该结点不再需要

3） 小心分析引用类型的指针

**六、 从尾到头打印链表**

题目：输入一个链表，从尾到头打印链表每个节点的值。

解决方案：

使用栈，从左到右将链表一一地存入栈当中，然后再将栈的值输出

## 队列

队列是一种特殊的线性表，只允许在表的前端front进行删除操作，在表的后端rear进行插入操作，和栈一样，队列是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为对头。

队列元素服从**先进先出**原则：FIFO——First In First Out

**题目1：拓扑排序（删除入度为0）保留，在图的地方在写**

对一个有向无环图G进行拓扑排序，是将G中所有顶点排成线性序列，使得图中任意一队顶点u和v，若边，则u在线性序列中出现在v之前。

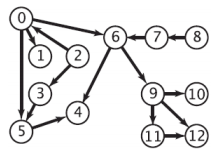
思路：

1）从有向图中选择一个没有前驱（即入度为0）的顶点并且输出它；

2）从图中删去该顶点，并且删去从该顶点发出的全部有向边；

3）重复上述两步，直到剩余的图中不再存在没有前驱的顶点为止

答案：2 🡪 8 🡪 0 🡪3 🡪 7 🡪1 🡪5 🡪 6 🡪9 🡪4 🡪11 🡪10 🡪12



**拓扑排序的进一步思考：**

1. 拓扑排序的本质是不断输出入度为0的点，该算法可用于判断图中是否存在环；

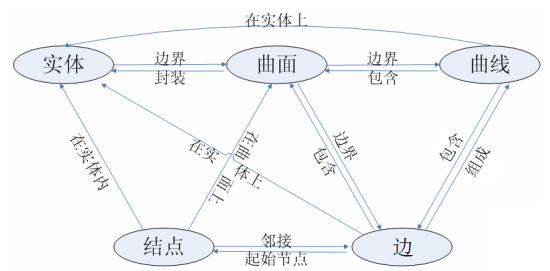
2. 可以用队列（或者栈）保存入度为0的点，避免每次遍历所有点，即每次更新连接点的入度即可；

3. 拓扑排序其实是给定了结点的一组偏序关系。

4. “拓扑”的涵义不限于此，在GIS中，它往往指点、线、面、体之间的相互邻接关系，即“橡皮泥集合”。存储这些关系，往往能够对某些算法带来好处。

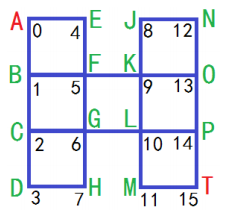
比如：计算不自交的空间曲面是否能够围成三维体

**扩展：拓扑排序的几何含义**



**题目2：最短路径条数问题（栈）图问题**

给定如图所示的无向连通图，假定图中所有边的权值都为1，显然，从源点A到终点T的最短路径有多条，求不同的最短路经的数目。



**问题分析：**

**求权值相同的最短路径问题，则原来的单源点Dijkstra算法退化成BFS的广度优先搜索**

**题目3：最长括号匹配**

给定字符串，仅包含左括号‘（’和右括号‘）’，它可能不是括号匹配的，设计算法，找出最长匹配的括号子串，返回该子串的长度。

如：

（（）：2

（）（）：4

（）（（））：6

（（）（））：6

算法分析：

1. 记起始匹配位置start=-1；最大匹配长度ml=0

2. 考察第i位字符c

3. 如果c为左括号，压栈；

4. 如果c为右括号，它一定与栈顶左括号匹配；

5. 如果栈为空，表示没有匹配的左括号，start=i，为下一次可能的匹配做准备

6. 如果栈不空，出栈（因为和c匹配了）；

7. 如果栈为空，i-start即为当前找到的匹配长度，检查i-start是否比ml更大，使得ml得以更新；

8. 如果栈不空，则当前栈顶元素t是上次匹配的最后位置，检查i-t是否比ml更大，使得ml得以更新。

附：进一步思考：

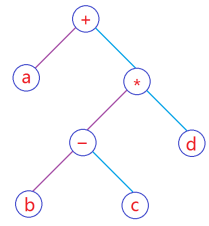
1. 经过分析算法得知，只有在右括号和左括号的发生匹配时，才有可能更新最终解；

2. 做记录前缀串p[0,…,i-1]中左括号数目与右括号数目的差x，若x为0时，考察是否最终解得以更新即可。这个差x，其实是入栈的数目，代码中用“深度”deep表达；

3. 由于可能出现左右括号不相等——尤其是左括号数目大于右括号数目，所以，再从右向前扫描一次。这样完成的代码，用deep值替换了stack栈，空间复杂度由O(N)降为O(1).

**题目4：逆波兰表达式RPN（Reverse Polish Notation）（栈）**

逆波兰表达式，又称为后缀表达式

事实上，二元运算的前提下，中缀表达式可以对应一颗二叉树；逆波兰表达式即该二叉树后序遍历的结果。

中缀表达式：a+(b-c)\*d

后缀表达式：abc-d\*+

计算给定的逆波兰表达式的值，有效操作只有+-\*/，每个操作数都是整数。

如：

“2”,”1”,”+”,”3”,”\*”: 9 ----- (2+1)\*3

“4”,”13”,”5”,”/”,”+”: 6 ----- 4+(13/5)

算法分析： 栈

1. 若当前字符是操作数，则压栈；

2. 若当前字符是操作符，则弹出栈中的两个操作数，计算后仍然压入栈中

3. 若某次操作，栈内无法弹出两个操作数，则表达式有误。

逆波兰表达式的用途

1. 计算数学表达式的最常用方法；（去除优先级处理

2. 在实践中，往往给出的不是立即数，而是变量名称；若经常计算且表达式本身不变，可以事先将**中缀表达式转换成逆波兰表达式存储**。

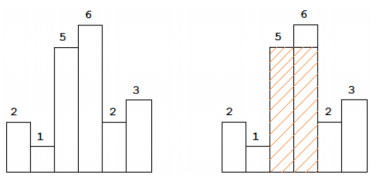
3.电脑的计算器用的就是逆波兰表达式

4. 复杂条件可以用逆波兰表达

比如 a + b > 0.7 or c > 0.5 🡺 ab+0.7>c0.5>or

**题目5：直方图矩形面积（栈）**

给定n个非负整数，表示直方图的方柱的高度，同时，每个方柱的宽度假定都为1；试找出直方图中最大的矩形面积。如：给定高度为：2,1,5,6,2,3，最大面积为10.



算法分析：

**方案一. 暴力解决**

1. 将直方图的数组记做a[0…size-1];

2. 计算以方柱a[i]为右边界的直方图中，遍历a[0…i]，依次计算可能的高度和面积，取最大者；

3. i从0遍历到size-1;

4.时间复杂度为O(N^2)

**方案二：利用特殊性质**

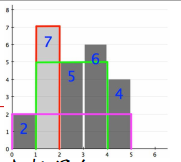
分析：若a[i+1] 》a[i]，则以a[i]为右边界的矩形Rect(width, height)，总可以添加a[i+1]带来的矩形Rect(1, height)，使得面积增大；只要当a[i+1] < a[i]时，才计算a[i]为右边界的矩形面积。（注：为了算法的一致性，在a[0…size-1]的最后，添加a[size]=0，保证a[size-1]为右边界的矩形得到计算）

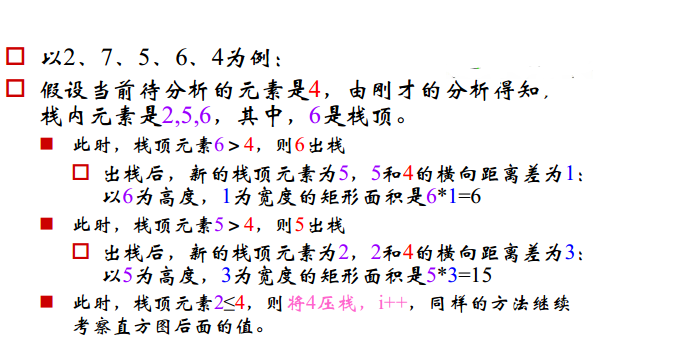
思路：

1. 从前向后遍历a[0…size]（末尾添加了0），若a[i]>a[i-1]，则将a[i]放入缓冲区；

2. 若a[i]《a[i-1]，则计算缓冲区中能够得到的最大矩形面积

例子：





为了能够方便的计算“横向距离”，压入栈的是方柱的索引，而非方柱的高度本身。

**题目6：收集雨水问题(待解决)**

给定n个非负整数，表示直方图的方柱的高度，同时，每个方柱的宽度假定都为1。若使用这样形状的容器收集雨水，可以盛多少水量？

如输入：0,1,0,2,1,0,1,3,2,1,2,1

返回6。

## 堆栈

1. 模拟队列，使用两个栈

### 摸拟队列（Stack\_use.java）

题目：用两个栈来实现一个队列，完成队列的Push和Pop操作。 队列中的元素为int类型。

解决方案：

实现队列的push，只需一个栈即可，将元素直接存入栈中。实现队列的pop操作，则需要两个栈，将一个栈里面的全部取出，然后存入另一个栈中，便可实现队列的先进先出的特点。

## 总结

1. 一般而言，栈的重要度大于队列

2. 栈的用途非常广泛，除了表达式求值，在深度优先遍历，保存现场等问题中常常出现

3. **思考：一个栈（无穷大）的进栈序列为1,2,3…,n，共多少种不同的出栈序列？（Catalan数）**

**解决方案：Catalan数**



C:\Users\Johnqiu\Desktop\equation.png

The first Catalan numbers for n = 0, 1, 2, 3, … are

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, … (sequence A000108 in the OEIS).

http://c.hiphotos.baidu.com/baike/s%3D257/sign=97158a763df33a879a6d071ff15d1018/2e2eb9389b504fc2974eb943e2dde71190ef6d66.jpg

http://d.hiphotos.baidu.com/baike/s%3D162/sign=f68c65f4b5b7d0a27fc9009bf9ee760d/5d6034a85edf8db190ab75220e23dd54574e74ea.jpg

康托编码

参考文献：

<https://en.wikipedia.org/wiki/Catalan_number>

http://blog.sina.com.cn/s/blog\_6917f47301010cno.html

# 第三讲：字符串

主要内容：

1. 字符串循环移位

2. LCS最长公共子序列

3. 字符串全排列

4. KMP模式串匹配

5. 字符串的最长回文字串

6. Manacher算法

7. BM算法

8. 替换空格

9. 找出数组中重复的数字

## 字符串循环左移

**题目：**给定一个字符串S[0…N-1]，要求把S的前K个字符移动到S的尾部，如把字符串“abcdef”前面的2个字符‘a’、‘b’移动到字符串的尾部，得到的新字符串“cdefab”：即字符串循环左移k。

**提示：**1.循环左移n+k位和k位的效果相同。

2.循环左移k位等价于循环右移n-k位。

**算法要求：**

时间复杂度为O(n)，空间复杂度为O(1)。

**算法分析：**

**方案一：暴力移位法**

每次循环左移1位，调用k次即可

时间复杂度为O(kN)，空间复杂度为O(1)

**方案二：三次拷贝**



时间复杂度为O(N)，空间复杂度为O(k)

**方案三：三次翻转**



例子：abcdef

X=ab X’=ba;

Y=cdef Y’=fedc

(X’Y’)’ = (bafedc)’ = cdefab

时间复杂度为O(N)，空间复杂度为O(1)

该问题会在“完美洗牌”算法中再次遇到

**问题：翻转的次序倒过来，先整体翻转，再局部翻转，方案是否可行？**

## LCS最长公共子序列

**LCS的相关定义：**

1. 最长公共子序列，即Longest Common Sequence， LCS。

2. 一个序列S任意删除若干个字符得到新序列T，则T叫做S的子序列；**（不要求连续，但要求顺序）**

3. 两个序列X和Y的公共子序列中，长度最长的那个，定义为X和Y的最长公共子序列；

比如：字符串13455与245576的最长公共子序列为455

字符串acdfg与adfe的最长公共子序列为adf

注意区别最长公共子串（Longest Common Substring）

最长公共字符串要去连续

**LCS的意义所在：**

1. 广泛应用在图形相似处理，媒体流的相似比较，计算生物学方面

2. 可以用来描述两段文字之间的“相似度”，即它们的雷同程度，从而能够用来辨别抄袭。另一方面，判断文字的修改的部分，往往十分准确。

**算法分析：**

方案一：暴力求解，穷举法

1. 假定字符串X，Y的长度分别为m，n；

2. X的一个子序列即下标序列{1,2,…,m}的严格递增子序列，因此，X共有2^m个不同子序列；同理，Y有2^n个不同子序列，从而穷举搜索法需要指数时间O(2^m \* 2^n)；

3. 对X的每一个子序列，检查它是否也是Y的子序列，从而确定它是否为X和Y的公共子序列，并且在检查过程中选出最长的公共子序列；

显然，不可取。

**LCS的相关记号：**

1. 有字符串X，长度为m，从1开始数；字符串Y，长度为n，从1开始数；

2.即X序列的前i个字符（）（Xi为“字符串X的i前缀”）

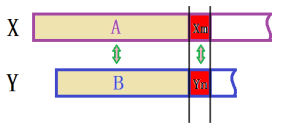
3. 即Y序列的前j个字符串（）（Yi为“字符串Y的j前缀”）

LCS(X,Y)为字符串X和Y的最长公共子序列，即为

算法分析：

假设（最后一个字符相同），则：与的最长公共子序列的最后一个字符必定为，那么，

解释如下：

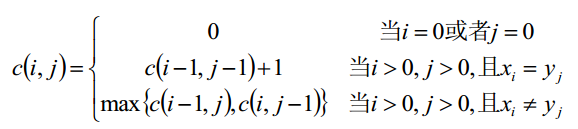


记，则W是的子序列；同理，W是的子序列；因此，W是和的公共子序列。

反证法：假设W不是和的最长公共子序列，不妨记，且，那么将W换成，得到更长的，与题设矛盾

设有二维数组c[m,n]，其中c[i,j]记录序列Xi和Yj的最长公共子序列的长度。那么

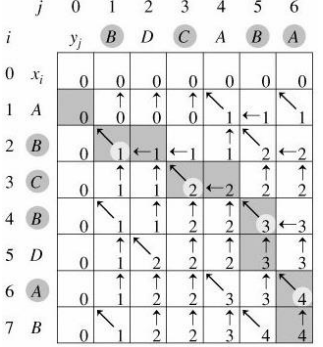
状态转移方程组：



例子：

X = <A,B,C,B,D,A,B>

Y = <B,D,C,A,B,A>



若需要把最长公共子序列打印出来：

|  |
| --- |
| i = s1len  j = s2len  s = []    # 逆序打印序列  while i!=0 and j!=0:  if s1[i-1] == s2[j-1]: # 相等直接相加  s.append(s1[i-1])  i -= 1  j -= 1  else:  if chess[i][j-1] > chess[i-1][j]: #由左边变化而来  j -= 1  else: # 由右边变化而来  i -= 1  s = s[::-1]  return s |

思考：如果只计算LCS的长度，直接使用滚动数组可以降低空间复杂度。

原因在于：无需存储m\*n的大数组，只需用一个滚动数组即可

## LIS最长递增子序列

LIS，全称为Longest Increasing Subsequence，定义为找出给定数组最长且单调递增的子序列。

如：给定数组{5,6,7,1,2,8}，则其最长的单调递增子序列为{5,6,7,8}，长度为4.

解决方案一：使用LCS

假设原数组为A{5,6,7,1,2,8}，排序后为A’{1,2,5,6,7,8}，

由于，原数组A的子序列顺序保持不变，而且排序后A’本身就是递增的，这样，就保证了两序列的最长公共子序列的递增特性。如此，若想求数组A的最长递增子序列，其实就是求数组A与它的排序数组A’的最长公共子序列。

解决方案二：动态规划

第n项最长子序列的长度，由前n项的最长子序列的最大值所决定

设置一个最长序列长度的数组longest[],前缀元素pre[]

1. 判断每一个元素的值是否比它前面的值大，若大，说明该元素可以添加到前面的值后面

## 字符串全排列

题目：给定字符串S[0…N-1]，设计算法，枚举S的全排列。

以字符串1234为例：

1-234, 2-134, 3-214, 4-231

## KMP模式串匹配

## 字符串的最长回文子串

## Manacher算法

## BM算法

## 替换空格（ReplaceSpace.java）

**题目：**请实现一个函数，把字符串中的每个空格替换成“%20”，例如输入“We are happy”，则输出“We%20are%20happy.”。

**算法分析：**

方案一：从头到尾扫描字符串，每一次碰到空格字符的时候，依次将空格后面所有的字符都后移两个字节，然后替换成%20。

举例：

将“We are happy.”换成“We%20are%20happy.”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| W | e |  | a | r | e |  | h | a | p | p | y | . | \0 |  |  |  |  |  |
| W | e | % | 2 | 0 | a | r | e |  | h | a | p | p | y | . | \0 |  |  |  |
| W | e | % | 2 | 0 | a | r | e | % | 2 | 0 | h | a | p | p | y | . | \0 |  |

黄色部分移动1 次，红色部分移动2次。

时间复杂度是O(n^2)

方案二：先统计出字符串中的所有的空格，求出新字符串所需要的长度，字符串的长度等于原来的长度加上2乘以空格数目，然后设置两个指针，第一个指针P1指向字符串的最后一个字符串，第二个指针P2指向新字符串的最后一个字符，从右往左扫描字符串，移动P1，逐个把它指向的字符复制到P2指向的位置，直到遇到空格为止。接着移动P2，将字符串“%20”插入相应的位置。

时间复杂度为O(n)

# 第四讲：数组

主要内容：

1. 求局部最大值

2. 第一个缺失的整数

3. 旋转数组的最小值

4. 寻找零子数组

5. 数组的最大间隔

6. 最大连续子数组

7. 荷兰国旗问题

8. Cantor数组

9. 子集和数问题

10. 数组中重复的数字

11. 构建乘积数组

12. 字符流中第一个不重复的字符

13. 调整数组顺序使奇数位于偶数前面

**核心思想：**

**循环不变式：如果某命题初始为真，且每次更改后仍然保持该命题为真，则若干次更改后改命题仍然为真**

## 求局部最大值(LocalMax)

**题目：给定一个无重复元素的数组A[0…N-1]，求找到一个该数组的局部最大值。**

规定：在数组边界外的值无穷小。即：A[0]>A[-1]，A[N-1]>A[N]。从而可得如下局部最大值的形式化定义：



**定义：**若子数组Array[from,…,to]满足：

Array[from] > Array[from-1]

Array[to] > Array[to + 1]

称该子数组为“高原数组”。若高原数组长度为1，则该高原数组的元素为局部最大值。

**举例：1,2,13,6,7,8,11,9,10**

**输出：11**

**解决方案：**

采用二分法，得到中间值，若中间值的后面一个位置比中间值大，则将high指针指向中间，否则将low指针指向mid+1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 13 | 6 | 7 | 8 | 11 | 9 | 10 |
| low |  |  |  | mid |  |  |  | high |
|  |  |  |  |  | low | mid |  | high |
|  |  |  |  |  | low, mid | high |  |  |
|  |  |  |  |  | mid | **low,** high |  |  |

**代码：**

|  |
| --- |
| /\*\*  \* Return one of the local max values in the array  \*  \* **@param** A a disorderly array  \* **@param** size the size of A  \* **@return** the local max  \*/  **public** **static** **int** findLocalMax(**int**[] A , **int** size){  **int** left = 0;  **int** right = size-1;  **int** mid;    **while**(left < right){  mid = (left + right) / 2;  **if** (A[mid] > A[mid + 1]){  right = mid;  }**else**{  left = mid + 1;  }  }  **return** A[left];  } |

## 第一个缺失的整数(FirstLostInteger)

**题目：**给定一个数组A[0,…,N-1]，找到从1开始，第一个不在数组中的正整数。

**例子：**输入：3,5,1,2，-3,7,14,8

**输出：**4

**解决方案：**

方案一：暴力搜索

从1开始搜索，判断是否在数组出现，直到某个数，在数组搜索不到为止，然后将该数输出

方案二：循环不等式

利用数组的下标，使数组从1开始计算：

若A[i] = i, i 加1，继续比较后面的元素。

若A[i] < i或 A[i] > N 或 A[A[i]] = A[i]，将A[N]赋值给A[i]，然后N减1.

若A[i] > i，则将A[A[i]和A[i] 交换

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 第0轮 | 0 | 3(i) | 5 | 1 | 2 | -3 | 7 | 14 | 8 |
| 第1轮 | 0 | 1 | 5(i) | 3 | 2 | -3 | 7 | 14 | 8 |
| 第2轮 | 0 | 1 | 8(-3)(i) | 3 | 2 | 5 | 7 | 14 |  |
| 第3轮 | 0 | 1 | 14(8)(i) | 3 | 2 | 5 | 7 |  |  |
| 第4轮 | 0 | 1 | 7(14)(i) | 3 | 2 | 5 |  |  |  |
| 第5轮 | 0 | 1 | 5(7)(i) | 3 | 2 |  |  |  |  |
| 第6轮 | 0 | 1 | 2(5)(i) | 3 |  |  |  |  |  |
| 第7轮 | 0 | 1 | 2 | 3(i) |  |  |  |  |  |
| 第8轮 | 0 | 1 | 2 | 3 | i |  |  |  |  |

**代码：**

|  |
| --- |
| /\*\*  \* Find the fist miss Integer of the array  \* **@param** a Array  \* **@param** size the size of array  \* **@return** the first miss number  \*/  **public** **static** **int** firstMissNumber(**int**[]a, **int** size){  **int**[] temp = **new** **int**[size+1];  **for** (**int** i=0; i < size; i++){  temp[i+1] = a[i];  }    **int** j = 1;  **while**(j <= size){  // three conditions  **if** (temp[j] == j){  j++;  }**else** **if**((temp[j] < j) || (temp[j] > size) || (temp[j] == temp[temp[j]])){  temp[j] = temp[size];  size--;  }**else**{ // temp[j] > j  // swap value  **int** tempV = temp[temp[j]];  temp[temp[j]] = temp[j];  temp[j] = tempV;  }    **for** (**int** k=0; k <= size; k++){  System.***out***.print(temp[k]+",");  }  System.***out***.println(" ");  }  **return** j;  } |

## 查找旋转数组的最小值(FindMinRotateArray)

**题目：**假定一递增排序数组以某个未知元素为支点做了旋转，如：原数组0124567旋转得到4567012。请找出旋转后数组的最小值。假定数组中没有重复数字。

**例子：**输入：4567012

输出：0

**解决方案：**



原数组为A，绕点旋转得到A1和A2（把前半部分看做A1，把后半部分看做A2）。需要寻找的目标点为C点（最小值），我们可以使用二分查找法。

首先，使用low和high求得中间点mid，分成两种情况处理：

若mid的值比high指向的值大（比如a点，它比右侧值大），那么low = mid + 1；

若mid的值比high指向的值小（比如b点，它比右侧值小），那么high = mid

这样不断地缩小范围，直到找到c点位置

**代码：**

|  |
| --- |
| **package** com.neu.Array;  /\*\*  \* The <tt>FindMinRotateArray</tt> Java provides a Binary search method for  \* finding the min Integer of the rotation array  \* <p>  \* <tt>Question:</tt>Given a rotation array A[0...N-1] , then  \* find the min Integer of the rotation array  \*  \* **@author** Johnqiu  \*/  **public** **class** FindMinRotateArray {    /\*\*  \* Find the min Integer of the rotation array  \* **@param** a Array  \* **@param** size the size of array  \* **@return** the first miss number  \*/  **public** **static** **int** findMin(**int**[]A, **int** size){  **int** low = 0;  **int** high = size-1;  **int** mid;    **while**(low < high){  mid = (low + high) /2;  **if** (A[mid] < A[high]){  high = mid;  }**else** **if**(A[mid] > A[high]){  low = mid +1;  }  }  **return** A[low];  }  /\*\*  \* Unit Test  \* **@param** args  \*/  **public** **static** **void** main(String[] args) {  **int**[] a = {4,5,6,7,0,1,2};  **int** m = *findMin*(a, a.length);  System.***out***.println(m);  }  } |

**题目2**：输入一个非递减排序的数组的一个旋转，输出旋转数组的最小元素。例如数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。NOTE：给出的所有元素都大于0，若数组大小为0，请返回0。

题目分析：

本题有两大陷阱：1. 非递减排序（有可能出现重复数字），比如2,2,2,2,2,2,1,2，意味着前面的方法需要进行改进，否则进入无限循环； 2. 数组大小为0时，返回0，

## 寻找零子数组(NSum)

**题目：**求对于长度为N的数组A，求连续子数组的和最接近0的值，比如：数组A：1,-2,3,10,-4,7,2,-5，它是所有子数组中，和最接近0的是哪个？

**例子：输入：**1,-2,3,10,-4,7,2,-5

输出：-4,,7,2,-5

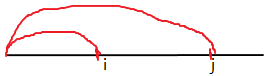
**解决方案：**

**方案一（不行就用它）：**暴力求解

先从最简单的入手，有总比没有强，遍历所有的子数组，时间复杂度为O(n^2)

**方案二：**借助前项和求解

比如我们求得前i项和是S[i]，前j项和是S[j]，如下图所示：



那么，S[j]-S[i]表示的从i到j之间的子数组的和。利用这一点，我们首先求得所有的前项和，接着对前项和按从小到大进行排序，然后相邻前项和进行相减，求得最接近0的子数组

**A：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **-2** | **3** | **10** | **-4** | **7** | **2** | **-5** |

**Sum：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **-1** | **2** | **12** | **8** | **15** | **17** | **12** |

**排序后的Sum：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **-1** | **0** | **1** | **2** | **8** | **12** | **12** | **15** | **17** |

最小的差值为：0， 位置是5到8

**代码：**

|  |
| --- |
|  |

## 数组的最大间隔(CalcMaxGap)

**题目：**给定整数数组A，求这N个数排序后最大间隔。如：1，7，14，9，4，13的最大间隔为4。

**例子：**输入：1,4,7,9,13,14

输出：13-9=4

**解决方案：**

方案一：对原数组进行排序，然后求后项减前项的最大值，即为解

方案二：借鉴桶排序/Hash映射的思想

假定N个数的最大最小值为max，min，则这N个数形成N-1个间隔，其最小值是，如果N个数完全均匀分布，则间距全部是且最小；如果N个数不是均匀分布，间距不均衡，间距必须大于；

思路：将N个数用间距分成N-1个区间，则落在同一区间内的数不可能有最大间距。统计后一区间的最小值与前一区间的最大值的差即可。（若没有任何数落在某区间，则该区间无效，不参与统计）。

**代码：**

|  |
| --- |
| /\*\*  \* Find the max gap of a array  \* **@param** a Array  \* **@param** size the size of array  \* **@return** the first miss number  \*/  **public** **static** **int** calcMaxGap(**int**[]A, **int** size){  TagSBucket[] tBuckets = **new** TagSBucket[size];  **for** (**int** i=0; i < size; i++){  TagSBucket tBucket = **new** TagSBucket();  tBuckets[i] = tBucket;  }  **int** nMax = A[0];  **int** nMin = A[0];  **int** i ;    // get max and min  **for** (i=0; i < size; i++){  **if** (nMax < A[i]){  nMax = A[i];  }**else** **if** (nMin > A[i]){  nMin = A[i];  }  }    //add num  **int** delta = nMax - nMin;  **int** nBucket;  **for** ( i = 0; i < size; i++){  nBucket = (A[i] - nMin) \* size / delta;  System.***out***.println(nBucket+"---nB");  System.***out***.println(A[i]+"--A[i]");  **if** (nBucket >= size){  nBucket = size -1;  }  tBuckets[nBucket].Add(A[i]);  }    // compute gap  i = 0;  **int** nGap = delta / size;  **int** gap;  **for** (**int** j=1; j < size; j++){  **if** (tBuckets[j].isbValid()){  gap = tBuckets[j].getnMin() - tBuckets[i].getnMax();  **if**(nGap < gap){  nGap = gap;  }  i = j;  }  }  **return** nGap;  } |

## 最大连续子数组(MaxSubArray)

**题目：**给定一个数组A，求A的连续子数组，使得该子数组的和最大（若需要输出子数组本身应该如何求解）

**例子：**

**输入：1,-2,3,10,-4,7,2,-5**

**输出：3,10,-4,7,2**

**解决方案：**

**方案一：按照零子数组的方法求**

首先求得所有的前项和，接着对前项和按从小到大进行排序，然后用最大项减去最小项，便可以得到最大连续子数组之和，然后再返过来获取他们的索引即可，

求得的时间复杂度(OlogN)

**A：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **-2** | **3** | **10** | **-4** | **7** | **2** | **-5** |

**Sum：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **-1** | **2** | **12** | **8** | **15** | **17** | **12** |

**排序后的Sum：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **-1** | **0** | **1** | **2** | **8** | **12** | **12** | **15** | **17** |

最大的差值为：18， 位置是3到7

**方案二：使用动态规划的方法求解**

从最后的一个项进行思考，由于它是连续的子数组，必须要经过当前项（与求最大的非连续子数组不同，它只考虑当前项要还是不要），所以只存在两种情况，要么将当前项加入子数组，要么从当前项开始计算。

定义子问题：子数组加入当前项，与当前项相比，哪个更大？（这里只要确定子数组和大于0即可）

定义状态：S[i]表示以A[i]结尾的数组中和最大的子数组

定义状态转移：S[i+1] = max(S[i] + A[i+1], A[i+1])

时间复杂度为O(n)

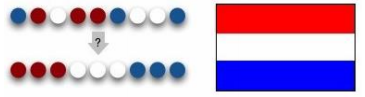
不采用DP数组，只用一个变量

**代码：**

|  |
| --- |
| /\*\*  \* Find the sum of sub array which is the biggest  \* **@param** a Array  \* **@param** size the size of array  \* **@return** the sum of subarray  \*/  **public** **static** **int** maxSubarray(**int**[]A, **int** size){    **if**(A.length == 0 || size ==0){ // illegal input  **return** -1;  }    **int** sum = A[0]; // begin from the first value  **int** result = sum; // the current best answer    **for** (**int** i=1; i < size; i++){  **if** (sum > 0){  sum += A[i];  }**else**{  sum = A[i];  }  result = Math.*max*(sum, result);  }  **return** result;  } |

## 荷兰国旗问题(HollandProblem)

**题目：**现有红、白、蓝三个不同颜色的小球，乱序排列在一起，请重新排列这些小球，使得红白蓝三色的同颜色的球在一起。（之所以叫荷兰国旗问题，是因为可以将红白蓝三色小球想象成条状物，有序排列后正好组成荷兰国旗）。



**例子：**

**输入：**012110022

**输出：000111222**

**解决方案：**

**方案一：原地调整，不创建空间**

使用三个指针，begin、current、end，begin指向的0，current指向的是1,，end指向的是2，

初始值：begin=0, current =0, end=N-1：

分成三种情况：

当A[cur] == 0时，则：

若begin == cur，则begin++, cur++

若begin!= cur， 则A[cur]与A[begin]交换，begin++, cur不变

若A[cur] == 1时，则：

cur++，begin不变，end不变

若A[cur] == 2时，则A[cur]与A[end]交换，end--, cur不变

改进版本：

由于和begin交换后，cur指向的一定为1，所以可以直接加1

当A[cur] == 0时，则：

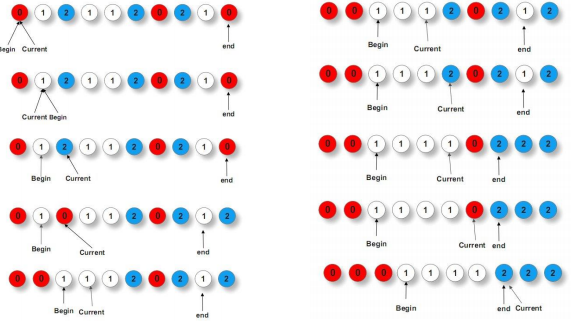
若begin == cur，则begin++, cur++

若begin!= cur， 则A[cur]与A[begin]交换，begin++, cur++

若A[cur] == 1时，则：

cur++，begin不变，end不变

若A[cur] == 2时，则A[cur]与A[end]交换，end--, cur不变



**代码：**

|  |
| --- |
| /\*\*  \* Solving Holland Problem  \* **@param** a Array  \* **@param** size the size of array  \* **@return** the first miss number  \*/  **public** **static** **int**[] holland(**int**[]A, **int** size){  **int** begin =0;  **int** current = 0;  **int** end = size -1;    **while**(current <=end){  **if**(A[current] == 2){ // A[current] == 2  **int** temp = A[end];  A[end] = A[current];  A[current] = temp;  end--;  }**else** **if**(A[current] == 1){  current++;  }**else**{  **if** (begin != current){  **int** temp = A[begin];  A[begin] = A[current];  A[current] = temp;  }  begin++;  current++;  }  }  **return** A;  } |

## Cantor数组(Cantor)

**题目：**已知数组A乱序着前N个正整数，现统计后缀数组A中小于元素A[i]的数目，并存放在数组C[i]中。如给定数组A={4,6,2,5,3,1}，得到数组C={3,4,1,2,1,0}。问：给定数组C={3,4,1,2,1,0}，如何恢复数组A？

**例子：**

给定数组A = {4,6,2,5,3,1}，那么：统计后缀数组小于A[i]的数目

{4:3，6:4，2:1，5:2，3:1，1:0}，即数组C={3,4,1,2,1,0}

**解决方案：**

方案一：（空间换时间）

1. 给定顺序数组B={1,2,3,…,N-1,N}，从0开始数
2. 循环Cantor数组C：

2.1 A[i] = B[C[i]]

2.2 在序列数组B中删除B[C[i]]

该方案的时间复杂度为O(N)，空间复杂度为O(N^2)

算法举例：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| C/A/B | 1 | 2 | 3 | 4 | 5 | 6 |
| 3 |  |  |  | 4 |  |  |
| 4 |  |  |  | X |  | 6 |
| 1 |  | 2 |  | X |  | X |
| 2 |  | X |  | X | 5 | X |
| 1 |  | X | 3 | X | X | X |
| 0 | 1 |  |  |  |  |  |

还原数组A：【4,6,2,5,3,1】

代码：

|  |
| --- |
| /\*\*  \* Resume cantor to the orginal array  \* **@param** a cantor array  \* **@param** b original array  \* **@param** size the size of array  \*/  **public** **static** **void** cantorResume(**int**[] a, **int**[]b, **int** size){  ArrayList<Integer> list =**new** ArrayList<Integer>();  **for** (**int** i=1; i <= size; i++){  list.add(i);  }  **for** (**int** j=0; j < size; j++){  **int** index = a[j];  **int** num = list.remove(index);  b[j] = num;  }    } |

方案二：降低空间复杂度，直接修改Cantor数组（时间换空间）

算法举例：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A/C | 3 | 4 | 1 | 2 | 1 | 0 |
| 1（第6位） | 2 | 3 | 0 | 1 | 0 | 0 |
| 2（第3位） | 1 | 2 | 0 | 0 | 0 | 0 |
| 3（第5位） | 0 | 1 | 0 | 0 | 0 | 0 |
| 4（第1位） | 0 | X |  | X | 5 | X |
| 5（第4位） |  | X | 3 | X | X | X |
| 6（第2位） | 1 |  |  |  |  |  |

Cantor数组中第一个出现0的位置：它表示位于该位置右侧的所有的元素都大于该元素，则该元素必然是最小的。每次找到第一个0后，将0左侧的Cantor值都减一，重复以上操作。

**代码：**

|  |
| --- |
| /\*\*  \* Resume cantor to the orginal array  \* **@param** a cantor array  \* **@param** b original array  \* **@param** size the size of array  \*/  **public** **static** **void** cantorResume2(**int**[] a, **int**[]b, **int** size){  **int** i,j;  **for**( i = 0; i < size ; i++){  **for** (j=0; j < size; j++){  **if**(b[j] !=0)  **continue**;  **if**(a[j] == 0)  **break**;  a[j]--;  }  System.***out***.println(j);  b[j] = i+ 1;  }  } |

## 子集和数问题

**题目：**已知数组A[0…N-1]，给定某数值sum，找出数组中若干个数，使得这些数的和为sum。

**例子：**

**解决方案：**

方案一：动态规划

**代码：**

## 数组中重复的数字（FindDuplicateInteger）

**题目：**在一个长度为n的数组里的所有数字都在0到n-1的范围内，数组中某些数字是重复的，但不知道有几个数字重复了，也不知道每个数字重复了几次，请找出数组中任意一个重复的数字。

例如，如果输入长度为7的数组{2,3,1,0,2,5,3}，那么对应的输出是重复的数字2或者3

**解决方案：**

方案一：排序

对数组进行排序，然后从头到尾扫描排序后的数组，将当前位置的数字和前一个位置的数字比较就可以了。

时间复杂度：O(nlogn)

方案二：哈希表

从头到尾按顺序扫描数组的每个数，每扫描到一个数字的时候，都可以用O(1)的时间来判断哈希表里是否已经包含了该数字，如果哈希表里还没有这个数字，就把它加入到哈希表里。如果哈希表里已经存在该数字了，那么就找到一个重复的数字。

时间复杂度：O(n)

空间复杂度：O(n)

方案三：循环不等式（要求数组的值 < 数组的长度）

从头到尾依次扫描这个数组中的数字，当扫描到下标i的数字时，首先比较这个数字（用m表示）是不是等于i。如果是，接着扫描下一个数字。如果不是，再拿它和第m个数字进行比较。如果它和第m个数字相等，就找到了一个重复的数字（该数字在下标为i和m的位置都出现了）。如果它和第m个数字不相等，就把第i个数字和第m个数字交换，把m放到属于它的位置。接下来再重复这个比较、交换的过程，直到我们发现一个重复的数字。

举例：数组{2,3,1,0,2,5,3}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 第0轮 | 2 | 3 | 1 | 0 | 2 | 5 | 3 |
| 第1轮 | 1(i) | 3 | 2 | 0 | 2 | 5 | 3 |
| 第2轮 | 3(i) | 1 | 2 | 0 | 2 | 5 | 3 |
| 第3轮 | 0(i) | 1 | 2 | 3 | 2 | 5 | 3 |
| 第4轮 | 0 | 1 | 2 | 3 | 2(i) | 5 | 3 |

得到重复数字为2

## 构建乘积数组（FindDuplicateInteger）

**题目：**给定一个数组A[0,1,…,n-1]，请构建一个数组B[0,1,…,n-1]，其中B中的元素B[i] = A[0]XA[1]X..A[i-1]XA[i+1]X..XA[n-1]，不能使用除法。

**解决方案：**

把B[i] = A[0]XA[1]X..A[i-1]XA[i+1]X..XA[n-1]分成两个部分A[0]XA[1]X..A[i-1]和A[i+1]X..XA[n-1]的乘积，不妨定义C[i] = A[0]XA[1]X..A[i-1] 和D[i] = A[i+1]X..XA[n-1]，形成矩阵B[i]，C[i]可以用自上而下的顺序计算出来，即C[i] = C[i-1] X A[i-1]，D[i]可以用自下而上的顺序计算出来，即D[i] = D[i+1] X A[i+1]

## 字符流中第一个不重复的字符（FindDuplicateInteger）

题目：请实现一个函数用来找出字符流中第一个只出现一次的字符。例如，当从字符流中只读出前两个字符“go”时，第一个只出现一次的字符是“g”。当从该字符流中读出前六个字符“google”时，第一只出现一次的字符是“l”。

解决方案：建立

## 调整数组顺序使奇数位于偶数前面（FindDuplicateInteger）

**题目：**输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有的奇数位于数组的前半部分，所有的偶数位于位于数组的后半部分，**并保证奇数和奇数，偶数和偶数之间的相对位置不变。**

解决方案：（类似于荷兰国旗问题）

,

# 第五讲：树

主要内容：

1. 二叉查找树（增删改查）

2. 前序中序后序遍历

3. 平衡二叉树

4. B树及其变种

5. R树

关键思路：递归、循环、栈、队列

## 二叉查找树

**题目：**写出二叉查找树的增删查改

二叉查找树是满足以下条件的二叉树：

1. 左子树上的所有结点值均小于根结点值
2. 右子树上的所有结点值均不小于根结点值
3. 左右子树也满足上述两个条件

二叉查找树的插入：

1. 若当前的二叉查树为空，则插入的元素为根结点，
2. 若插入的元素值小于根结点值，则将元素插入到左子树中，
3. 若插入的元素值不小于根结点值，则将元素插入到右子树中，
4. 递归上述过程，直到找到插入点为叶子结点

二叉查找树的删除：

记待删除的结点为P，分三种情况进行处理：

1. P为叶子结点。

直接删除该结点，再修改P的父结点的指针

1. P为单支结点（即只有左子树或右子树）。

将P的子树与P的父亲节点相连，删除P即可

1. P的左子树和右子树均不空。

找到P结点的中序前驱或后继替代P所指结点，然后再从原排序二叉树中删去中序前驱或后继结点。（也就是用大于P的最小结点或小于P的最大结点来代替P结点）

**题目：**给定前序遍历序列和中序遍历序列，求后序遍历

**解决方案：**

1. 根据前序遍历和中序遍历建立二叉树
2. 对二叉树进行后序遍历数组

## 平衡二叉树

**平衡二叉树（AVL树），其特点为：**

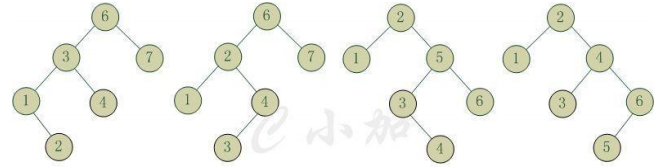
1. 每个结点的左右子树的高度之差不超过1
2. 如果插入或者删除结点后高度差大于1，则进行结点旋转，重新维护平衡状态。

**解决了二叉查找树退化成链表的问题，即从根结点开始往左依次递减，或者从根结点开始往右依次递增**。

**分析高度不平衡结点**

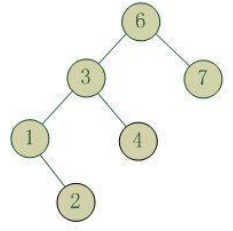
维护不平衡的直接手段是选择不平衡结点X的某孩子C作为父结点，当前结点X作为原子结点C的子结点。这显然会影响到X的子结点、孙结点，因此，分成如下四种情况：

高度不平衡结点的两颗子树的高度差2



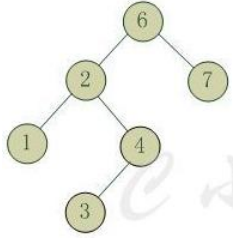
高度不平衡1：**左左**

6结点的左子树3结点高度比右子树7结点大2，左子树3结点的左子树1结点高度大于右子树4结点，这种情况称为左左。



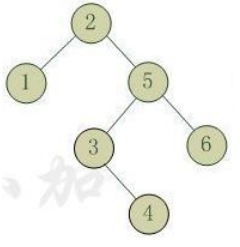
高度不平衡2：**左右**

6结点的左子树2结点高度比右子树7结点大2，左子树2结点的左子树1结点高度小于右子树4结点，这种情况称为左右。



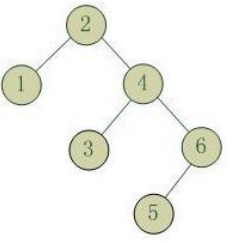
高度不平衡3：**右左**

2结点的左子树1结点高度比右子树5结点小2，右子树5结点的左子树3结点高度大于右子树6结点，这种情况称为右左。



高度不平衡4：**右右**

2结点的左子树1结点高度比右子树4结点小2，右子树4结点的左子树3结点高度小于右子树6结点，这种情况称为右右。

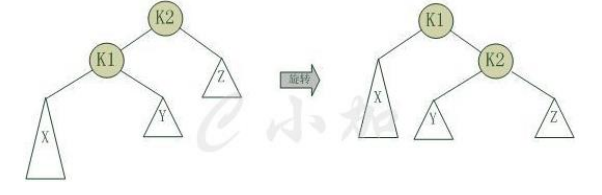


**左左和右右对称；左右和右左对称；**

左左和右右两种情况是对称的，这两种情况的旋转算法是一致的，只需要经过一次旋转就可以达到目标，称之为单旋转。

左右和右左两种情况是对称的，这两种情况的旋转算法是一致的，需要进行两次旋转就可以达到目标，称之为双旋转。

**单旋转：**结点K2不满足平衡特性，因为它的左子树K1比右子树Z深2层，而且K1子树中，更深的一层的是K1的左子树X子树，所以属于左左情况。



假设K2不平衡：中序遍历X-K1-Y-K2-Z

1. 把K1作为新的根结点
2. K2>K1，所以，把K2置于K1的右子树上
3. K2>Y>K1，所以，把Y置于K2的左子树上

这样的操作只需要一部分指针改变，结果我们得到另外一颗二叉查找树，它是一棵AVL树，因为X向上一移动了一层，Y还停留在原来的层面上，Z向下移动了一层。整棵树的新高度和之前没有在左子树上插入的高度相同，插入操作使得X高度长高了。因此，由于这颗子树高度没有变化，所以通往根节点的路径就不需要继续旋转了。

**双旋转：**

对于左右和右左这两种情况，单旋转不能使它达到一个平衡状态，要经过两次旋转。双旋转是针对于这两种情况的解决方案，同样的，这样两种情况也是对称的，只要解决了左右这种情况，右左就很好办了。图4是左右情况的解决方案，节点k3不满足平衡特性，因为它的左子树k1比右子树Z深2层，而且k1子树中，更深的一层的是k1的右子树k2子树，所以属于左右情况。



为使树恢复平衡，我们需要进行两步，第一步，把k1作为根，进行一次右右旋转，旋转之后就变成了左左情况，所以第二步再进行一次左左旋转，最后得到了一棵以k2为根的平衡二叉树。

左右情况 —（右右旋转）—> 左左情况 —（左左旋转）—> ALV树

**删除：**

删除的方法也和二叉查找树的一致，区别是，删除完成后，需要从删除节点的父亲开始向上维护树的平衡一直到根节点

相关模块：

|  |
| --- |
| private:         TreeNode<T>\* root;//根节点         void insertpri(TreeNode<T>\* &node,T x);//插入         TreeNode<T>\* findpri(TreeNode<T>\* node,T x);//查找         void insubtree(TreeNode<T>\* node);//中序遍历         void Deletepri(TreeNode<T>\* &node,T x);//删除         int height(TreeNode<T>\* node);//求树的高度         void SingRotateLeft(TreeNode<T>\* &k2);//左左情况下的旋转         void SingRotateRight(TreeNode<T>\* &k2);//右右情况下的旋转         void DoubleRotateLR(TreeNode<T>\* &k3);//左右情况下的旋转         void DoubleRotateRL(TreeNode<T>\* &k3);//右左情况下的旋转         int Max(int cmpa,int cmpb);//求最大值   public:         AVLTree():root(NULL){}         void insert(T x);//插入接口         TreeNode<T>\* find(T x);//查找接口         void Delete(T x);//删除接口         void traversal();//遍历接口 |

## B树

M阶B树需要满足的条件：

1. 每个结点至多有m个孩子；
2. 除根结点外，其他结点至少有m/2个孩子；
3. 根结点至少有2个孩子；
4. 所有叶结点在同一层；
5. 有个孩子的非叶结点有个关键字；结点内部，关键字递增排列

# 第六讲：图

主要内容：

-1 .图的拓扑排序

0. 图的建立以及增删改查、遍历

1. 并查集

2. 图的存储

3. 最短路径（Dijkstra/Floyd/Bellman-Ford）

4. 最小生成树（MST） Prim/Krusal

5. 图的搜索（广度优先搜索 / 深度优先搜索/ 动态规划）

6. 单词变换问题 / 周围区域问题

7．括号匹配的字符串 / 八皇后问题 / 数独问题

8. 分割词汇问题

9. 马踏棋盘

## 图的拓扑排序

**拓扑排序（删除入度为0）保留，在图的地方在写**

对一个有向无环图G进行拓扑排序，是将G中所有顶点排成线性序列，使得图中任意一队顶点u和v，若边，则u在线性序列中出现在v之前。

思路：

1）从有向图中选择一个没有前驱（即入度为0）的顶点并且输出它；

2）从图中删去该顶点，并且删去从该顶点发出的全部有向边；

3）重复上述两步，直到剩余的图中不再存在没有前驱的顶点为止

答案：2 🡪 8 🡪 0 🡪3 🡪 7 🡪1 🡪5 🡪 6 🡪9 🡪4 🡪11 🡪10 🡪12

## 并查集

问题：动态连通性

输入：n个整数

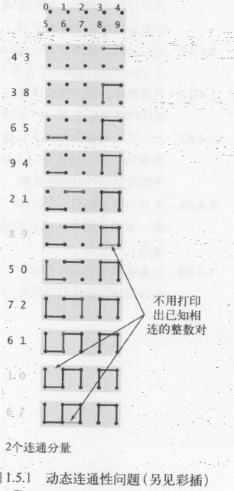
多个整数对（p和q）

输出：若p和q不相连，输出，并添加进入连通集，若p和q相连，则不输出。

“相连”具备对等性：

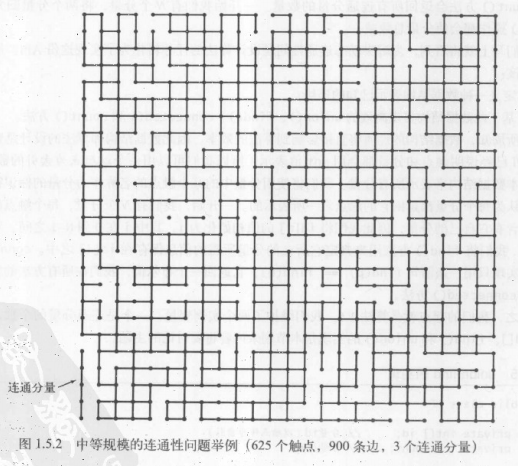
1. 自反性：p和p是相连的；
2. 对称性：如果p和q是相连的，那么q和p也是相连的；
3. 传递性：如果p和q是相连的且q和r是相连的，那么p和r也是相连的。

举例：



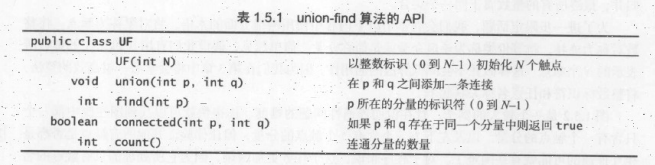
相关应用：网路、变量名等价性、数学集合

难度分析：



该问题具有625个点，900条边，求任意给点两个点的连通性

相关的API如下：



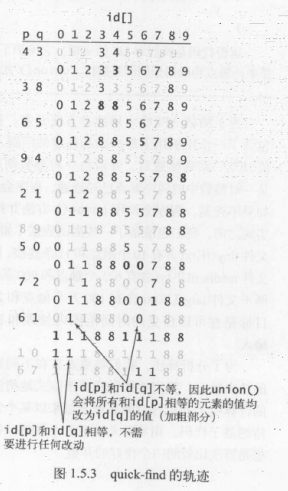
**解决方案**：

**方案一：quick-find算法**

思想：当且仅当id[p]等于id[q]时，p和q是连通的，换句话说，在同一个连通分量中的所有触点在id[]中的值必须相同。

Union算法步骤如下：

1. 获取p和q在id数组中的值pid和qid
2. 若pid等于qid，说明p和q已经在同一个分量下，不采取任何操作；若不相等，说明两者不在同一个分量下，循环整个id数组，将数组中值等于pid的位置的值更新为qid，然后分量数量减1；



最坏时间复杂度：O(N^2)

缺陷：

不能运用于大型问题，因为对于每一对输入union()都需要扫描整个id[]数组

**方案二：quick-union算法**

使用id数组来存储每个结点的父亲结点，用父链接的形式表示了一片森林。

思路：从给定的结点开始，由它的链接得到它的父亲结点，再由父亲结点找到下一个父亲结点，如此循环，直到找到根结点，即链接指向自己的结点

**方案三：加权quick-union算法**

思路：记录每一课树的大小并总是将较小的树连接到较大的树上。

# 第七讲：图实践

# 第八讲：查找排序

主要内容：

1. 归并排序 / 逆序对（完成）
2. 杨氏矩阵的增删改查（完成）
3. Gantt图
4. 2-sum问题（完成）
5. 素和阶数问题（完成）
6. 排序本身：插入排序（完成）、选择排序（完成）、希尔排序（完成）、冒泡排序（完成）、堆排序（完成）、快速排序（完成）、记数排序（完成）、桶排序（完成）、基数排序（完成）、锦标赛排序

7．选择前K个大的数问题

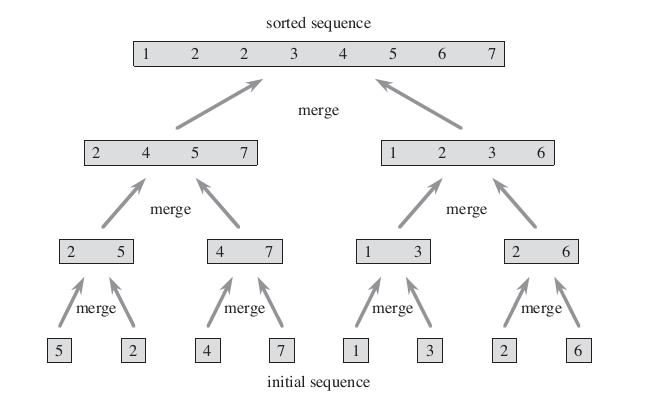
## 归并排序

**思路：**将原始序列看成n个只含有一个元素的序列，然后两两归并，形成若干个有序二元组，再两两归并，形成若干有序四元组……依次类推，最后只剩下两个子序列，再进行一次归并，便完成整个排序

**例子：**

原始序列：5、2、4、7、1、3、2、6

排序后序列：1、2、2、3、4、5、6、7



**代码：**

|  |
| --- |
| void MergeSort (int R[], int l ,int r){  if (l < r){  // 找出中间索引  int center = (l + r) / 2;  mergeSort(R, l, center); // 左边数组排序  mergeSort(R, center+1, r); // 右边数组排序  merge(R, l, center, r); // 合并  }  }  /\* 任意两个数组进行合并\*/  void merge(int R[], int l, int center, int r){  int[] tempR = new int[R.length];  int rbegin = center + 1; // 右边数组第一位  int tmpIndex = l; // 临时数组索引  int newIndex = l; // 新数组索引  while(l <= center && rbegin <= r){  // 获取两个数组最小值放入临时数组  if(R[l] < R[rbegin]){  tempR[tmpIndex++] = R[l++];  }else{  tempR[tmpIndex++] = R[rbegin++];  }  }  // 将剩余的数放入临时数组  while(rbegin <= r){  tempR[tmpIndex++] = R[rbegin++];  }  while(l <= center){  tempR[tmpIndex++] = R[l++];  }  // 更新原数组  while(newIndex <= r){  R[newIndex] = tempR[newIndex];  newIndex++;  }  } |

**时间复杂度：**

选取merge函数内的“归并操作”作为基本操作

总共需要进行log2n趟排序，每趟排序执行n次基本操作，所以

f(n) = nlog2n

T(n) = O(nlog2n)

与初始序列无关，即最好情况为O(nlog2n)，最坏情况为O(nlog2n)，平均情况为O(nlog2n)

**空间复杂度：**

T(n) = O(n)

## 逆序数问题（有问题）

**问题：**给定一个数组A[0…N-1]，若对于某两个元素a[i]，a[j]，若i<j且a[i] > a[j]，则称(a[i],a[j])为逆序对。一个数组中包含的逆序对的数目称为该数组的逆序数。试设计算法，求一个数组的逆序数。

**举例：**3, 56, 2, 7的逆序数为3

（3,2），（56,2），（56,7）

**解决方案**：

使用归并算法进行排序并进行统计。

## 杨氏矩阵

**问题：**给定MXN的二维数组，每一行、每一列都是有序的，则该二维数组称为杨氏矩阵

相关操作：

1. **增（Insert）：**

思路：

1）找到插入点的合适位置：

先按行查找，然后在按列查找。

若遇到比自己小的位置，则停止，

否则，比较行查找和列查找的值，选择最大的那个值交换

2）交换找寻到的位置

举例：

输入：9, 16, 3, 2, 4, 8, 5, 14, 12

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 12 | max | max | max |
| max | max | max | max |

插入7：

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 12 | max | max | max |
| max | max | max | 7 |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 12 | max | max | 7 |
| max | max | max | max |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 12 | max | 7 | max |
| max | max | max | max |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 12 | 7 | max | max |
| max | max | max | max |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 7 | 12 | max | max |
| max | max | max | max |

**2．查找（Find）**

思路：

从表格的最右上角出发，一直往下寻找。如果找到该值，那么直接输出，否则如果表格值大于该值，那么向前一列寻找（c--），如果表格值小于该值，那么向后一行寻找（r++）

举例：

输入：9, 16, 3, 2, 4, 8, 5, 14, 12

寻找 12

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 12 | max | max | max |
| max | max | max | max |

先从14开始，由于14大于12，并且14还是第四列的第一个数字，因此12不可能出现在数字14所在的列，于是我们把这一列从需要考虑的区域内剔除，之后只需要分析剩下的3列。在剩下的矩阵中，右上角为5,12比5大，说明12可能在5的右边或者下边，由于右边已经被排除，因此12只能在5的下边，一直走到max，12比max小，说明12可能在该行，然后不断地往左走，直至找到12.

**3．删除（delete）**

思路：

1. 将要删除的元素设置为∞。
2. 再通过将∞的这个元素向友和向下调整，最终移到矩阵满足条件为止。

举例：

输入：9, 16, 3, 2, 4, 8, 5, 14, 12,7

删除2

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 7 | 12 | max | max |
| max | max | max | max |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 3 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 7 | 12 | max | max |
| max | max | max | max |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 5 | 5 | 14 |
| 4 | 8 | 9 | 16 |
| 7 | 12 | max | max |
| max | max | max | max |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 5 | 9 | 14 |
| 4 | 8 | 9 | 16 |
| 7 | 12 | max | max |
| max | max | max | max |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 3 | 9 | 14 |
| 4 | 8 | 16 | 16 |
| 7 | 12 | max | max |
| max | max | max | max |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 5 | 9 | 14 |
| 4 | 8 | 16 | max |
| 7 | 12 | max | max |
| max | max | max | max |

## Gantt图

**问题：**军训中，小明和小强分别负责细君、昭君、探春、文成四人的站军姿和踢正步科目。根据军训要求，只有在学会站军姿之后才能进行踢正步训练，由于四人的天资差别，学习时间如下表。问：应该如何安排四官的学习时间，能够使得所有人学会这两项技能的时间最短？



思想：所有事件中的最短时间是昭君踢正步（1小时），由于该事件是第二阶段，则将其最后执行；此短时间是探春站军姿（2小时）

## 寻找和为定值的两个数

**问题：**输入一个数字A[0…N-1]和一个数字Sum，在数组中查找两个数，使得。

**解决方案：**

方案一：暴力求解

从数组中任意选取两个数x,y，判定它们的和是否为输入的数字Sum。时间复杂度为O(N^2)，空间复杂度为O(1)

方案二：两头扫

如果数组是无序的，先排序O(NlogN)，然后用两个指针i,j，各自指向数组的首尾两端，令i=0，j=n-1，然后i++，j--，逐次判断a[i]+a[j]是否等于Sum:

1. 若a[i] + a[j] > sum, 则i不变，j--；
2. 若a[i] + a[j] < sum, 则i++，j不变；
3. 若a[i] + a[j] == sum, 如果只要求输出一个结果，则退出；否则，输出结果后i++，j--;

数组无序的时候，时间复杂度最终为：

O(NlogN + N) = O(NlogN)

方案三：建立hash表

建立以数组的值为键和值的hash表，然后循环整个数组，判断以value-a[i]为键的值是否存在，若存在，则成功。

## 素和阶数

**题目：**一个正整数可以被拆分成两素和的数目为“素和阶数”。请计算100万以内哪个数的素和阶数最大。即求100万以内能被拆分成两素数之和的数目最多的一个数。

**解决方案：**

1. 使用Eratosthenes筛法计算100万以内所有素数，存储在数组P[0…size-1]中；
2. 正整数Z从1到100万次依次遍历：
3. 查找数组p中大于等于Z的最小值，记该最小值的位置为s
4. 使用2-Sum算法计算P[0,s-1]中和为s的所有组合
5. 遍历过程中，记录拆分种类最多的值。

**举例：**

寻找10以内能对拆分成两素数和最多的项

第一步：形成10以内的素数表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| false | false | true | true | false | true | false | true | false | false | false |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 5 | 7 |

第二步：循环10以内的数

0,1排除，从2开始，无法拆分两素和，到3，也不行，到4，可取素数为2,3，不行，到5，可取素数2,3，可以拆分成2+3，只有1对，到6，可取素数为2,3,5，不行，到7，可取素数为2,3，5,7，可以拆分成2+5，只要1对，到8，可取素数为2,3，5,7，可以拆分成3+5，只有1对，到9，可取素数为2,3，5,7，可以拆分成2+7，只有1对，到10，可取素数为2,3，5,7，可以拆分为3+7，只有1对

## 排序的稳定性

如果排序的过程中，只有相邻元素进行比较，是稳定的，如冒泡排序、归并排序；如果间隔元素进行了比较，往往是非稳定的，如堆排序、快速排序。

如果能够方便整理数据，对于不稳定的排序，可以使用（A[i],i）键对来进行算法，可以使得不稳定排序变成稳定排序

## 堆排序

大顶堆：父亲大孩子小

小顶堆：父亲小孩子大

（孩子之间的大小无要求）

思想：

1. 建堆

将原始序列建成完全树（只要位置对应即可，数字大小无限制），然后从右至左，从下至上地处理非叶子结点（a），若a的值小于孩子结点，则从中选出最大（小）的一个与a交换，则从中选出最大（小）的一个与a交换，直至建成大顶堆（小顶堆）；

2. 排序

将根结点（即当前无序序列中第一个元素）与最后一个元素交换，形成有序序列和无序序列；

3. 剩下的部分重复1的步骤，直至无序序列中的元素只剩下1个，排序结束

举例:

输入数组：4,1,3,16,9,10,14,8,7

1. 建立大顶堆：（树状形式）

（数组形式）

非子结点的位置：n/2-1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 1 | 3 | 16 | 9 | 10 | 14 | 8 | 7 |
| 4 | 1 | 3 | 16 | 9 | 10 | 14 | 8 | 7 |
| 4 | 1 | 14 | 16 | 9 | 10 | 3 | 8 | 7 |
| 4 | 16 | 14 | 8 | 9 | 10 | 3 | 1 | 7 |
| 16 | 9 | 14 | 8 | 4 | 10 | 3 | 1 | 7 |

2. 将16与最后一个数7进行交换，形成有序序列和无序序列

7,9,14,8,4,10,3,1,16

3. 继续将无序序列进行排序，直至无序序列中只剩下1个元素

## 选择前K个大的数问题

**问题：**给定N个数（N非常大，大数据），求出前K个大的数（大数据问题）

**解决方案：**

一般思路是，将N个数进行排序，然后取前K个数，但如果N个数是几十亿个数，无法加载到内存，该如何计算？

方案一：小顶堆

**只需维护一个大小为K的数据结构**，可以选择小顶堆。

具体流程如下：

1. 建立一个小顶堆，小顶堆的大小为k

2. for 每个数：

If 这个数比小顶堆的堆顶元素大：

弹出小顶堆的最小元素；

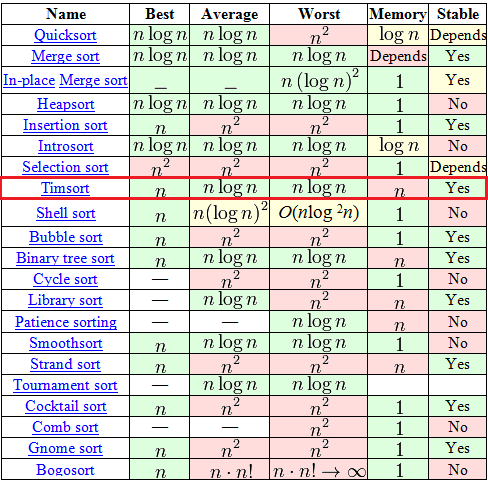
把这个数插入到小顶堆；

3. 小顶堆中的k个元素就是所要求的元素

小顶堆的作用：

方案二：大顶堆

方案三：BFPRT算法



Heap VS Quick：

快速排序的最直接竞争者是堆排序。堆排序通常比快速排序稍微慢，但是最坏情况的运行时间总是O(n log n).快速排序是经常比较快，但仍然有最坏情况性能的机会。

堆排序拥有重要的特点：仅使用固定额外的空间，即堆排序是原地排序，而快速排序需要O(log n)的空间

# 第九讲：动态规划

主要内容：

1. 消费金额（滴滴笔试题）

## 一、 消费金额

题目：某餐厅有n张桌子，每张桌子有一个参数：a可容纳的最大人数；有m批客人，每批客人有两个参数b人数，c预计消费金额。在不允许拼桌的情况下，请实现一个算法选择其中一部分客人，使得总预计消费金额最大

输入：

输入包括m+2行。第一行两个整数n（1<=n<=50000）,m（1 <= m <= 50000），第二行为n个参数a，即每个桌子可容纳的最大人数，以空格分隔，范围均在32位int范围内。接下来m行，每行两个参数b,c。分别表示第i批客人的人数和预计消费金额，以空格分隔，范围均在32位int范围内。

输出：

输出一个整数，表示最大的总预计消费金额

例子：

3 5

2 4 2

1 3

3 5

3 7

5 9

1 10

输出：

20

解决方案：

本题的关键点在于“不允许拼桌”，意味着一批客人和一张桌子是一一对应关系，一张桌子不能有两批客人，一批客人不能有两张桌子，因此可以使用贪心算法来求，只要保证每张桌子所能容许的客人的消费金额最大即可。

时间复杂度是O（mn）

|  |
| --- |
| **public** **class** Consumption {    /\*\*  \* create inner class comsumer  \* **@author** Johnqiu  \*  \*/  **class** Comsumer{  **int** num;  **int** money;    **public** Comsumer(**int** n, **int** v) {  // **TODO** Auto-generated constructor stub  **this**.num = n;  **this**.money = v;  }    @Override  **public** String toString() {  // **TODO** Auto-generated method stub  **return** **this**.money + "";  }  }    /\*\*  \* Comsumer Comparator  \* **@author** Johnqiu  \*  \*/  **class** ComsumerComparator **implements** Comparator<Comsumer>{  @Override  **public** **int** compare(Comsumer com1, Comsumer com2) {  // **TODO** Auto-generated method stub  **return** com1.money - com2.money;  }  }    **public** **static** **void** main(String[] args) {  Consumption con = **new** Consumption();  Scanner cin = **new** Scanner(System.***in***);  **int** n = cin.nextInt(); // table number  **int** m = cin.nextInt(); // people number  **int**[] tables = **new** **int**[n];  ArrayList<Comsumer> comsumers = **new** ArrayList<Comsumer>();  **for**(**int** i=0; i < n; i++){  tables[i] = cin.nextInt();  }  **for**(**int** j=0; j < m; j++){  **int** num = cin.nextInt();  **int** money = cin.nextInt();  Consumption.Comsumer c = con.**new** Comsumer(num, money);  comsumers.add(c);  }    Arrays.*sort*(tables); // sort tables  Collections.*sort*(comsumers, con.**new** ComsumerComparator()); // sort comsumers    **long** count = 0L;  **for**(**int** i=m-1; i >=0; i--){  **int** j = 0;  **for**(; j < n; j++){  **if**(tables[j] >= comsumers.get(i).num){  **break**;  }  }  **if**(j < n){  count += comsumers.get(i).money;  tables[j] = 0;  }  }  System.***out***.println(count);  }  } |

# 第十讲：概率组合海量数据处理

主要内容：

## 第一节 统计数字概率

题目：给定某正整数N，统计从1到N!的所有数中，首位数字出现1的概率。进而，可以计算首位数字是2的概率，是3的概率，从而得到一条**“九点分布”**。

思路：

1. 创建一个1到9的列表

2. 循环到正整数N：

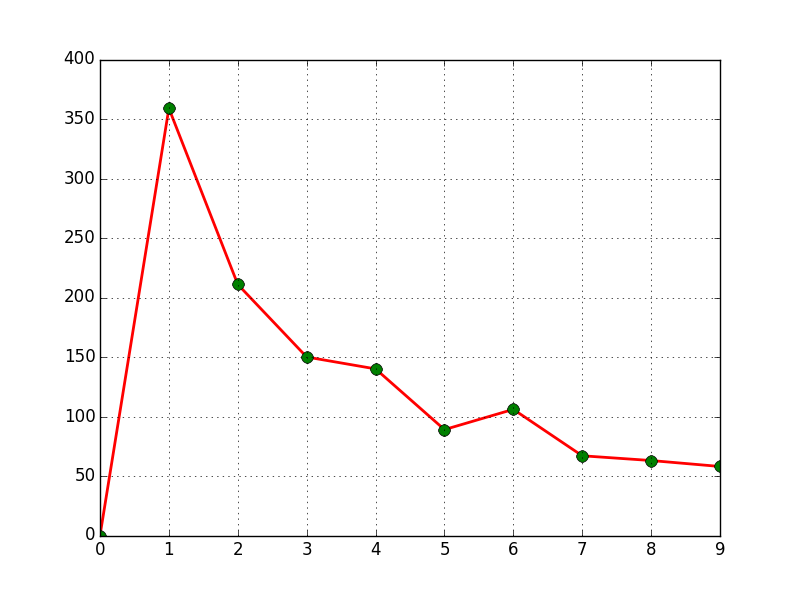
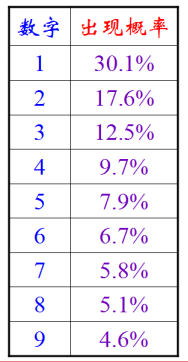
k = k \* i

获取k的首位数字

使用列表统计该数字

代码：

|  |
| --- |
| import matplotlib.pyplot as plt  class calculation\_firstNum(object):  def firstNum(self, num):  while (num / 10):  num = num / 10  return num    def paint(self, frequency):  plt.clf()  plt.plot(frequency,"r-",linewidth=2)  plt.plot(frequency,"go",markersize=8)  plt.grid(True)  plt.show()    if \_\_name\_\_=="\_\_main\_\_":  calc = calculation\_firstNum()  frequency = [0 for x in range(0,10)]  n = int(raw\_input("pleas input a num:"))  num = 1  for i in range(1,n+1):  num = num \* i  m = calc.firstNum(num)  frequency[m] = frequency[m] + 1  calc.paint(frequency) |



## 第二节 本福特定律

**本福特定律**（本福德法则，Frank Benford），又称第一数字定律，是指在实际生活得出的一组数据中，以1为首位数字出现的概率约为总数的三成；是直观现象1/9的三倍。推广来说，越大的数，以它为首几位的数出现的机率就越低，它可用于检查各种数据是否有造假。

该定律的公式为：F(d) = log[1 + (1/d)]（d为自然数）

相关应用： 阶乘 / 素数数列 / 斐波那契数列首位、住宅地址号码、经济数据反欺诈、选举投票反欺诈

定律使用范围：

1）这些数据必须跨度足够大，必须横跨好几个数量级才能产生这个结果。

2）有人为规则的数据就不满足该定律，比如手机号码、身份证号、发票编号等数据，明显不满足这种对数分布律。也就是说，本福特定律正是没有任何限制才显露处理的定律，越是对数据的产生有人为限制，越是不满足该定律。

3）数据不能经过人为修饰，随便人为修改的数据一般就不满足本福特定律

## 第三节 古典概型与几何概型

（古典概型）题目：麻将去除花牌后的标准麻将，由1到9的“万，条，饼”各4张，以及“东南西北中发白”各4张，共计136张组成。我们把两张内容一样的牌叫一副“将”，“东南西北”叫做“风牌”。请问，庄家起手摸14张牌，则他起手没有“将”的概率是多少？此外，可以算下摸13张牌没有“将”的概率，摸13张牌没有“风”的概率。

（几何概型）题目：A、B两国元首相约在首都机场晚20点至24点交换一份重要文件。如果A国的飞机先到，A会等待1个小时；如果B国的飞机先到了，B会等待2个小时。假设两架飞机在20点至24点降落机场的概率是均匀分布，

## 第四节 Buffon's Needle

题目：桌面上有距离为a的若干平行线，将长度为L的针随机丢在桌面上，则这根针与平行线相交的概率是多少？（假定L<a）

## 第五节 Gale-Shapley算法（稳定婚姻算法）

题目：婚介所登记了N位男孩和N位女孩，每个男孩都对N个女孩的喜欢程度做了排序，每个女孩都对N个男孩的喜欢程度做了排序。你作为月老，能否给出稳定的牵手方案？

## 第六节 猜数字游戏

题目：两个聪明人A和B玩猜数字的游戏。他们在脑门上各贴一个正整数数字，两个数字只相差1，A和B只能看到对方的数组而看不到自己的。

以下是两人的对话：

A：我不知道

B：我也不知道

A：我知道了

B：我也知道了

上述4句对话结束后，聪明的你帮助A、B推算下，他们的数字各是多少呢？

## 第七节 概率化商品推荐

题目：假设在某推荐场景中，经计算A和B两个商品与当前访问用户的匹配度分别为0.8分和0.2分，系统将随机为A生成一个均匀分布于0到0.8的最终得分，为B生成一个均匀分布于0到0.2的最终得分，试计算最终B的分数大于A的分数的概率。

## 第八节 圆内均匀取点

题目：给定定点O(x0,y0)和半径r，使得二维随机点(x,y)等概率落在圆内。

## 第九节 带拒绝的采样

题目：给定函数rand7()随机返回自然数1-7，利用rand7()构造随机返回1-10的函数rand10().

## 第十节 带权推荐

题目：假设歌曲库中N首歌，每首歌给定一个整数分数。现在要求从N首歌中随机选择若干首推荐给用户，要求推荐的这些歌是和其分数作为正比的。

## 第十一节 金钗赠诗问题

题目1：赛诗会后，十二金钗待奔前程，分别宴上，12人各写一首诗放入宝囊。大家任取，若取到自己的诗，则再取一首并放回自己的诗。12人都拿到别人的诗作算一种分配。问：共有多少种不同的分配？

题目2：赛诗会上十二金钗各赋诗一首，12人各自随机挑选一首后，李纨曰：“大家通过两两交换的方式，换回自己的律诗；但要求只能跟我交换”。现已知：

黛玉、宝钗、元春、探春、湘云、妙玉、迎春、惜春、熙凤、巧姐、李纨、可卿

各自拿到的律诗作者为：

熙凤、黛玉、迎春、惜春、湘云、可卿、探春、元春、宝钗、巧姐、妙玉、李纨

试计算至少需要多少次交换，才能使得所有人交换得到自己的律诗？

## 第十一节 滴滴海量ip地址转换

题目：在滴滴的大数据分析任务中经常会遇到根据用户IP地址查询用户归属地问题，现在有个文件source.txt，其中包括了n行ip地址（例如：114.246.68.141），有另一个文件ip\_dict.txt，里面包含了m行不同ip段到归属地的映射关系（例如：114.246.0.0/18 北京），ip段之间不重合（多个ip段可能对应相同的归属地）。请设计一个算法，要尽可能的将source.txt中的全部ip地址转换成“ip归属地”形式，并给出数据结构和复杂度分析。

# 第十一讲：智力题

主要内容：

1. 斐波那契数列

2. 青蛙跳阶

3. 变态青蛙跳阶

4. 矩形覆盖

5. 位运算

6. 数值的整数次方

7. 幸运数

8. 最佳位置

9. 爬山

10. 进制均值

## 一、 斐波那契数列

**题目**：大家都知道斐波那契数列，现在要求输入一个整数n，请你输出斐波那契数列的第n项。n<=39



**解决方案**：

**方案一：直接按公式递归**

问题：重复计算太多，时间复杂度过大

**方案二：记忆数组（解决递归重复的好帮手）**

由于重复计算的结果太多，因此我们可以将中间项保存起来，在下次计算的时候查找一下，如果已经计算过就不用再重复计算了。本题中，每一项的计算，只与该项的前两项相关，所以，我们只需设两个临时参数保存前两项的结果即可。

采用迭代的方法，从头开始计算。

例如：

F(2) = F(0) + F(1)

F(3) = F(1) + F(2)

F(4) = F(2) + F(3)

F(5) = F(3) + F(4)

时间复杂度为O(n)

**方案三：矩阵乘法**

数学公式：



只需求得，便可以求得f(n)。

问题转换成如何计算？

第一种：循环

从0开始循环到n-1，计算，时间复杂度为O(n)

第二种：递归



总结：用不同的方法求解斐波那契数列的时间效率大不相同。第一种基于递归的解法虽然直观但时间效率很低，在实际软件开发中不会用这种方法，也不可能得到面试官的青睐。第二种方法把递归的算法用循环实现，极大地提高了时间效率，第三种方法把求斐波那契数列转换成求矩阵的乘方（不常用）

## 二、 青蛙跳阶

**题目**：一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

**解决方案：**

**方案一：记忆数组（解决递归重复的好帮手）**

由于重复计算的结果太多，因此我们可以将中间项保存起来，在下次计算的时候查找一下，如果已经计算过就不用再重复计算了。本题中，每一项的计算，只与该项的前两项相关，所以，我们只需设两个临时参数保存前两项的结果即可。

## 三、 变态青蛙跳阶

**题目：**一只青蛙一次可以跳上1级台阶，也可以跳上2级……它也可以跳上n级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

解决方案：

**方案一：记忆数组**

推导公式可知，



创建一个长度为n+1的dp数组，用来记录每个计算过的f(n)，然后循环计算f(n)

|  |
| --- |
| **int**[] dp = **new** **int**[target+1];  dp[0] = 0;  dp[1] = 1;  dp[2] = 2;    **for**(**int** i = 3; i <= target; i++){  **int** temp = 0;  **for**(**int** j = i-1; j > 0; j--){  temp = temp + dp[j];  }  dp[i] = temp + 1;  } |

时间复杂度达到O(n^2),空间复杂度也达到O(n),

**方案二：**推导公式



两个公式进行相减，可得：



**方案三：**位操作

左移一位相当于该数乘以2，左移2位相当于该数乘以2^2=4。上面举的例子15<< 2=60，即乘了4。但此结论只适用于该数左移时被溢出舍弃的高位中不包含1的情况。

因此

## 四、 矩形覆盖（同斐波那契数列）

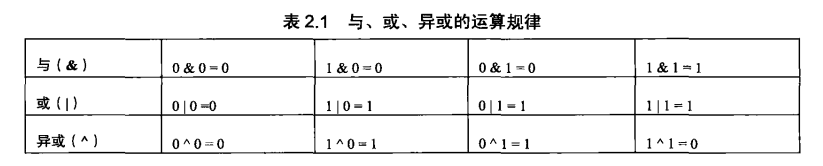
**题目：**我们可以用2\*1的小矩形横着或者竖着去覆盖更大的矩形。请问用n个2\*1的小矩形无重叠地覆盖一个2\*n的大矩形，总共有多少种方法？

解决方案：

令f(n) = 2\*n, 当n=1时，f(1)=1, 当n=2，f(2)=2， 当n=3时，f(3)=f(2)+f(1) = 3，通过分析可知，这仍然属于斐波那契数列。

## 五、 位运算

位运算只有5种运算：与、或、异或、左移、右移



左移运算符m<<n表示把m左移n位。左移n为的时候，最左边的n位将被丢弃，同时在最右边补上n个0，比如：

00001010 << 2 = 00101000

10001010 << 3 = 01010000

右移运算符m >> n 表示把m右移n位。右移n位的时候，最右边的n位将被丢弃。但右移时处理最左边位的情形要稍微复杂一点。如果数字是一个无符号数值，则用0填补最左边的n位。如果数字是一个有符号数值，则用数字的符号为填补最左边的n位，也就是说如果数字原先是一个正数，则右移之后在最左边补n个0；如果数字原先是负数，则友谊之后在最左边补n个1。比如：

00001010 >> 2 = 00000010

10001010 >> 3 = 11110001

**题目**：输入一个整数，输出该数二进制表示中1的个数。其中负数用补码表示。

**解决方案**：

把整数右移一位和把整数除以2在数学上是等价的，但是除法的效率比移位运算要低得多，在实际编程中应尽可能地**用移位运算符代替乘除法**。

如果一个整数与1做与运算的结果是1，表示该整数最右边一位是1，否则是0.

把一个整数减去1，再和原整数做与运算，会把该整数最右边一个1变为0，那么一个整数的二进制表示中有多少个1，就可以进行多少次这样的操作。

比如，1100，减去1后结果为1011，再将1100与1011做位与运算，得到的结果是1000，相当于我们把1100最右边的1变成了0，结果刚好就是1000。

## 六、 数值的整数次方（Power.java）

**题目**：给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方。

**解决方案**：

问题1. 处理base为0.0、exponent<0的情况，因为0的负数次方不能出现，所以该情况应该直接返回无效，即-1.0；

问题2. 处理exponent<0的情况。base的负数次方的值等于1/base的负数次方;

问题3. 处理double型数值相等问题。可以通过计算两个double型的数值之间的差值的精度，如果该精度在我们误差的允许范围，我们认为该两个double型的数值相等。

## 七、 幸运数

**题目**：小明同学学习了不同的进制之后，拿起了一些数字做起了游戏，小明同学知道，在日常生活中我们最常用的是是十进制数，而在计算机上，二进制数也很常见，现在对于一个数字。小明同学定义出了两个函数f(x) 和 g(x)。

f(x)表示把这个数用十进制写出后各个数位上的数字之和，如f(123)=1+2+3=6。g(x)表示把x这个数用二进制写出后各个数位上的数字之和。如123的二进制表示为1111011，那么g(123)=1+1+1+1+0+1+1=6。

小明同学发现对于一些正整数x满足f(x)=g(x)，他把这种数字称为幸运数，现在知道，小于等于n的幸运数有多少个。

输入：第一行一个整数T（T<=10000）表示数据组数，每组数据输入一个数（n<=100000）。

输出：每组数据输出一行，小于等于n的幸运数个数

样例：

3

1

5

21

输出：

1

1

3

提示：小于等于21的三个幸运分别为：1,20,21

限时：3000MS

空间：589824KB

## 八、最佳位置

题目：小B热衷于体育运动赛事活动，这样她就可以发挥自己的一项特长——赛事解说。小B因为自身的高颜值，加之评论犀利到位，语言诙谐，因此拥有一大批粉丝。今天的赛事在在三个圆形的运动场同时举行，小B希望能够在此次的赛事解说中再展风姿，占据最佳的观察点不止一个，她希望首选具有最大视角的观察点。小B对寻找观察点不是很在行，她希望请你帮忙寻找最合适的观察点，需要指出的是，运动场馆都是露天的，解说员的视线不受遮挡。

输入：输入中有多组测试，每组测试数据包含3行，每行代表一个运动场，每行的格式为x,y,r，其中（x,y）为运动场中心的坐标，r为运动场的半径，x <= 1000, y<=1000, 1 <= r<=1000，所有的数据均为正数，运动场互不相交（无公共点且其中心点不在同一直线上）。

输出：对每组测试数据，若存在最佳观察点，则在单独的一行里输出改点的坐标（保留6位小数），否则输出“No”

样例输入：

0 0 10

60 0 10

30 30 10

样例输出：

30.000000 0.00000

## 九、爬山

题目：小B喜欢爬山，他总是选择最平坦的路径，并记录每天的行程情况及达到的最高海拔，使得连续两天之间的海拔之差最多为一个单位，不幸的是，在行程结束时，他不小心掉进河里，造成部分记录信息遗失。他想知道自己行程中可能达到的最高海拔，你是否能帮忙？

输入：输入若干组，每组的第一行为空格分隔的两个整数n和m，1<=n<=10^8，1<=m<=10^5，分别表示行程天数以及未遗失的记录数，随后紧跟m行，每行为空格分隔的两个整数d和h，

1<=d<=n，0<=h<=10^8，表示行程的第几天及当天达到的最高海拔。

输出：对每组的输入，如果记录是可能的，则在单独的行中输出可能达到的最高海拔，否则输出字符串“IMPOOSIBLE”（不含引号）

样例输入：

8.2

2.0

7.0

8.3

2.0

7.0

8.3

样例输出：

2

IMPOSSIBLE

提示：第一天和最后一天的海拔可以是任何值

## 十、进制均值

题目：一个数A如果按2到A-1进制表达时，各个位数之和的均值是多少？所有的计算均基于十进制进行，结果也用十进制表示为不可约简的分数形式。

输入：输入中有多组测试数据，每组测试数据为一个整数A(1<=A<=5000)

输出：对每组测试数据，在单独的行中以X/Y的形式输出结果

样例输入：

5

3

样例输出：

7/3

2/1

# 第十二讲：位运算

# 第十一讲：笔试面试

面试流程：通常一轮面试是从面试官对照着简历了解应聘者的项目经历及掌握的技能开始的。在介绍自己的项目经历时，应聘者可以参照STAR模型，着重介绍自己完成的工作（包括基于什么平台，用了哪些技术，实现了哪些算法等），以及最终对项目组的贡献。

接着进入重头戏技术面试环节，在这一环节中面试官会从编程语言、数据结构和算法等方面考查应聘者的基础知识是否扎实全面，并且很有可能会要求应聘者编程实现一两个函数。如果碰到的面试题很简单，应聘者也不能掉以轻心，一定要从基本功能、边界条件和错误处理等方面确保代码的完整性和鲁棒性。如果碰到的题目很难，应聘者可以尝试画图让抽象的问题变得形象化，也可以尝试举几个具体的例子去分析隐含的规律，还可以尝试把大的问题分解成两个或者多个小问题再递归地解决小问题。这3种方法能够帮组应聘者形成清晰的思路，从而解决复杂的难题，很多面试题都不止一种解决方案，应聘者可以从时间复杂度和空间复杂度两个方面选择最优的解法。在面试过程中，面试官除了关注应聘者的编程能力外，他还会关注应聘者的沟通能力和学习能力，并有可能考查应聘者的知识迁移能力、抽象建模能力和发散思维能力。

在面试结束前的几分钟，面试官会给应聘者机会问几个最感兴趣的问题。应聘者可以从当前招聘的项目及其团队等方面提出几个问题。不建议应聘者在技术面试的时候向面试官询问薪资情况，或者立即打听面试结果。

——《剑指offer》

越是简单的问题，越是需要全面地考虑所有的情况。