记忆的两个点：1. 伪代码（思想）2. 代码中要注意的点

不用加减乘除的加法：

- 有位运算解决https://blog.csdn.net/lrs1353281004/article/details/87192205

排序

快速排序：

- 伪代码：定一个基准，设置 2. 分区操作：向左遍历，小于基准的放于基准的左边，向右大于基准的放于右边（交替遍历）3. 分治的思想，递归基准左右两边执行第二步。
- 注意的点：时间复杂度的计算，平均复杂度为 O(nlogn)，最差$O(n^2)$
- 实现时注意的点：1. 注意high，low，start，end之间的关系 2. 先右向左，high--，mid=num[low]，反之相反 3. 注意两重循环 4， 递归时low+1和low-1

归并排序：

- 伪代码：1. 从mid开始，逐层递归拆分新的数组，直到单独的数开始返回 2. 对数进行逐层返回递归合并操作。
- 注意的点：1. 先拆分再合并 3. 合并的部分三个while

堆排序：

- **完全二叉树** 是 一种除了最后一层之外的其他每一层都被完全填充，并且所有结点都保持向左对齐的树，向左对齐指的是（**堆是一种完全二叉树**）
- 伪代码：（以大顶堆为例）1. 构建大顶堆 （从右往左，从下往上调整，第一步找到两个子节点中较大值的下表j，第二步如果该节点i小于子节点j，交换两个节点）2. 将大根堆的堆顶节点(最大值)和最低最右的值交换，交换后除去最低最右的点（也就是最大点），剩余的点再构建大顶堆 3. 重复第二步，知道大顶堆的长度为1（也就是最小值），返回。

  > 参考：https://www.jianshu.com/p/d174f1862601

- 注意的点：1. 需要一个辅助空间，因为树是从1开始计数，而数组是从0开始计数 2. 在调整树节点有两层，一个是从右往左，从下往上（循环即可），一个是如果要调整的节点的子树不是叶子节点，需要一直往叶子节点处调整。 3. 一开始从非叶子节点开始调整，交换后只调整根节点 4. 需要注意判断条件

链表

剑指offer（3）：从尾到头打印链表,按链表从尾到头的顺序返回一个ArrayList

- 伪代码1非递归： 遍历链表，同时用一个列表储存链表的值

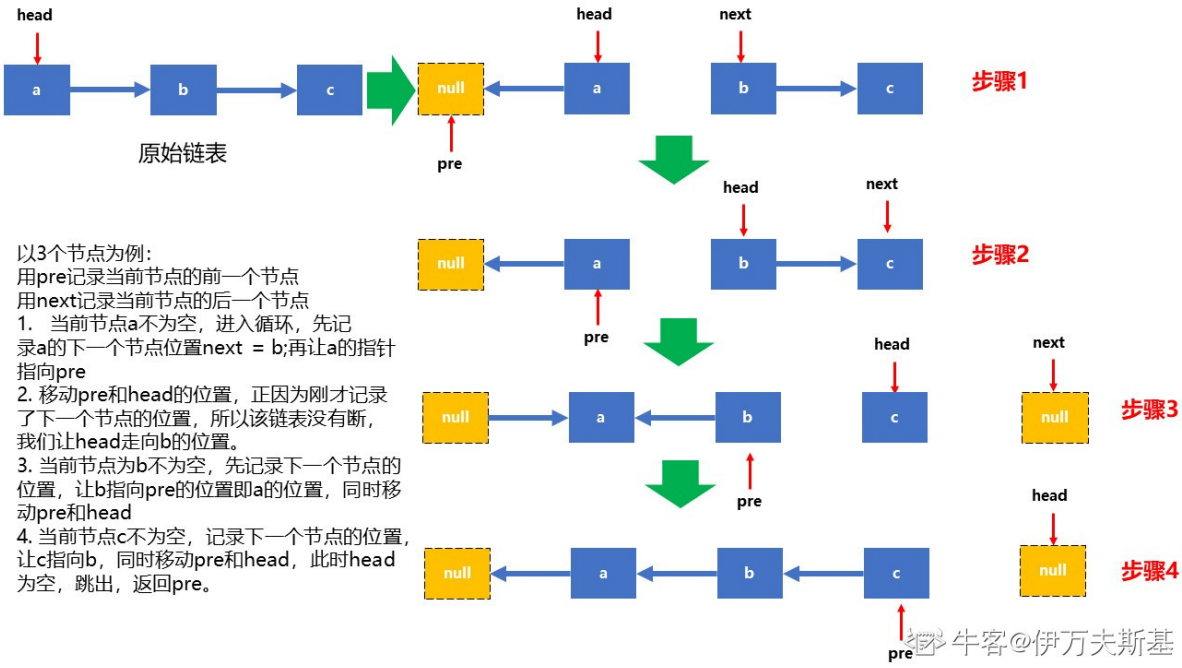
- 伪代码2递归： 递归逐层遍历链表，在递归返回的时候储存每个节点值（形成逆序）

剑指offer（14）：输入一个链表，输出该链表中倒数第k个结点

- 伪代码：1. 设置两个指针，p1，p2，先让p2走k-1步，然后再一起走，直到p2为最后一个 时，p1即为倒数第k个节点
- 注意的点： 注意在while循环的时候，判断条件

剑指offer（15）：反转链表，输入一个链表，反转链表后，输出新链表的表头。

- 伪代码：1. 三指针原地逆置法，见下图：



- 注意的点：逆序双向链表同样重要。

剑指offer（16）：合并两个有序的链表，重建一个新的，包含所有的两个链表。-->单调不减

- 伪代码： 1. 按照归并排序合并的思路 2. 设置两个指针 pHead1 和 pHead2，若均没遍历完：pHead1.val <= pHead2.val，那么当前 node 的 next 指向 pHead1。并且移动 pHead1 指针。反之相反，移动 node 指针。循环第二步，直到 3. 结束循环：如果 pHead1 未遍历完，node 的 next 指向 pHead1。如果 pHead2 未遍历玩，node 的 next 指向 pHead2

剑指offer（25）：复杂链表的复制

- 概念：复杂链表指的是每个节点中有节点值，以及两个指针，一个指向下一个节点，另一个特殊指针指向任意一个节点
- 伪代码：1. 复制每个节点，如：复制节点A得到A1，将A1插入节点A后面 2. 遍历链表，A1->random = A->random->next; 3. 将链表拆分成原链表和复制后的链表。
- 注意的点：三步分三个循环

具体分为三步：

（1）在旧链表中创建新链表，此时不处理新链表的兄弟结点

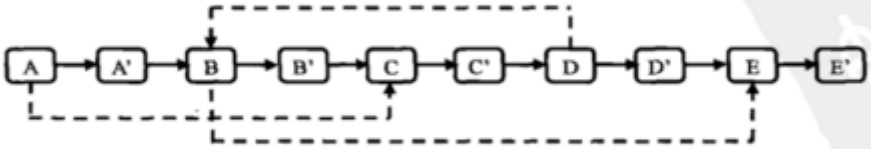


图 4.9　复制复杂链表的第一步

（2）根据旧链表的兄弟结点，初始化新链表的兄弟结点

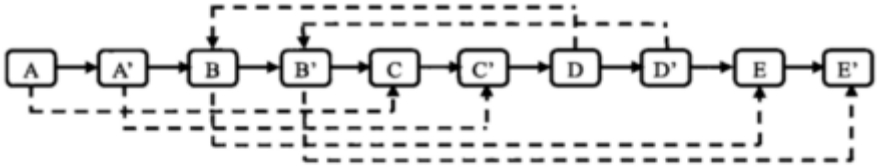


图 4.10　复制复杂链表的第二步

（3）从旧链表中拆分得到新链表

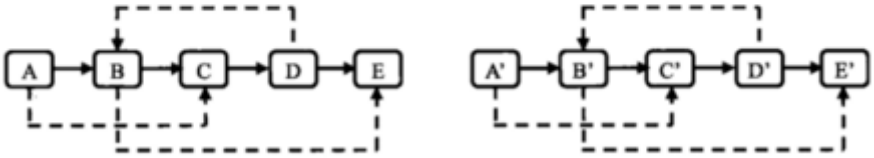


图 4.11　复制复杂链表的第三步

剑指offer（36）：输入两个链表，找出它们的第一个公共结点。

- 伪代码：注意，公共节点表示值和next指针都一样1. 找出2个链表的长度，然后让长的先走两个链表的长度差，然后再一起走（因为2个链表用公共的尾部）

剑指offer（55）：给一个链表，若其中包含环，请找出该链表的环的入口结点，否则，输出null。

- 伪代码：1. 快慢指针。快指针一次走两步，慢指针一次走一步，设链表起点到入口结点的长度是x1，快慢指针第一次相遇时距离入口结点的长度是x2，此时慢指针走了x1+x2，快指针走了2x1+2x2，也就是说x1+x2的长度正好是环的一圈大小的倍数。2. 此时让一个指针从起点出发，一个指针从相遇结点出发，都是一次走一步，当两个指针第一次相遇时恰好是在入口结点。

剑指offer（56）：删除链表中重复的结点

- 伪代码：1. 借助辅助头结点，可避免单独讨论头结点的情况。设置两个结点 pre 和 cur，当 cur 和 cur.next 值相等，cur 一直向前走，直到不等退出循环，这时候 cur 指的值还是重复值，调整 cur 和 pre 的指针再次判断、

数组

剑指offer（1）：二维数组查找，从左到右递增，从上到下递增

- 伪代码: 1. 矩阵是有序的,从左下角来看，向上数字递减，向右数字递增，重要思想。2. 从左下角开始查找，当要查找数字比左下角数字大时。右移。要查找数字比左下角数字小时，上移，知道找到

剑指offer（6）：非减排序的数组旋转之后，找到旋转数组的最小数字

把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。

- 伪代码：二分法的变种 1. 设置low，high，mid表示收尾和中间数，2. 比较low和high的数，如果low数小于high，直接返回low数 3. 比较low和mid的数，如果mid数大于low数（递增），low=mid+1 4. 重新得到mid，比较mid和high，如果mid小于high数（同递增），high=mid 5. 重复第234步，直到low>=high，返回 low数就是最小值
- 注意的点：**遇顺序数组，用二分**，2. 左边有序取另一半，右边有序右边取最小，这个是和二分查找不一样的地方，（判断条件改变）

剑指offer（13）：调整数组顺序使得奇数位于偶数之前。

- 伪代码：用插入排序的思想，时间复杂度最差O(n的平方)：1. `i++`往前走碰到偶数停下来，`j = i+1`，2. 若 `a[j]`为偶数，`j++`前进，直到碰到奇数。 3. 然后奇数插到a[i]位置，`j`经过的`j-i`个偶数依次后移 4. `j==len-1`时还没碰到奇数，证明`i`和`j`之间都为偶数了，完成整个移动

剑指offer（19）：顺时针打印矩阵

- 伪代码： 1. 定义四个变量代表范围边界，up、down、left、right，2. 向右走存入整行的值，当存入后，该行再也不会被遍历，代表上边界的 up 加一，同时判断是否和代表下边界的 down 交错 3. 向下走存入整列的值，当存入后，该列再也不会被遍历，代表右边界的 right 减一，同时判断是否和代表左边界的 left 交错。 4. 向左走存入整行的值，当存入后，该行再也不会被遍历，代表下边界的 down 减一，同时判断是否和代表上边界的 up 交错 5. 向上走存入整列的值，当存入后，该列再也不会被遍历，代表左边界的 left 加一，同时判断是否和代表右边界的 right 交错
- 注意的点：

剑指offer（28）：数组中出现次数超过一半的数字

- 伪代码：1. 对数组排序，找数组中间的数。2. 对这个中间数进行计数，判断是否超过一般、
- 注意的点，

剑指offer（29）：输入n个整数，找出其中最小的K个数

- 伪代码：1. 创建一个大小为k的数组，遍历n个整数，如果遍历到的数小于大小为k的数组的最大值，则将此数与其最大值替换。2. 数组用最大堆保存这k个数，每次遍历只和堆顶比，如果比堆顶小，删除堆顶，新数入堆。
- 注意的点：1. 这题可以用所有排序算法实现，2. O(N) 利用利用快速排序中的**获取分割（中轴）点位置函数getPartitiion**，基于数组的第k个数字来调整，使得比第k个数字小的所有数字都位于数

组的左边，比第k个数字大的所有数字都位于数组的右边。调整之后，位于数组左边的k个数字就是最小的k个数字（这k个数字不一定是排序的）但是会修改输入数组，且一般也不易想到。 3. O(N logK) 利用大顶堆，这个适合

剑指offer（30）：连续子数组（序列）的最大和

- 伪代码：动态规划 1. 最大子数组的和一定是由当前元素和之前最大连续子数组的和叠加在一起形成的，因此需要遍历n个元素，看看当前元素和其之前的最大连续子数组的和能够创造新的最大值。2. 具体步骤：dp[n]代表以当前元素为截止点的连续子序列的最大和，如果dp[n-1]>0，dp[n]=dp[n]+dp[n-1]，因为当前数字加上一个正数一定会变大；如果dp[n-1]<0，dp[n]不变，因为当前数字加上一个负数一定会变小。使用一个变量max记录最大的dp值返回即可。
- 注意的点： 也可以利用两个数值来记录，max_sum记录各个子序列的最大值，res记录当前序列+1后是否更大的值

剑指offer（32）：把数组排成最小的数

- 伪代码：就是一个排序的过程 1. 先将数组中每个元素转换成String类型，然后进行排序，如果str(a) + str(b) > str(b) + str(a),说明ab > ba，应该把b排在a前面。使用冒泡排序即可

剑指offer（35）： 输入一个数组,求出这个数组中的逆序对的总数

- 概念：在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。
- 伪代码： 归并排序的改进 1. 把数据分成前后两个数组(递归分到每个数组仅有一个数据项)，合并数组，合并时，出现前面的数组值array[i]大于后面数组值array[j]时；则前面数组array[i]~array[mid]都是大于array[j]的，count += mid+1 - i

剑指offer（37）：统计一个数字在排序数组中出现的次数。

- 伪代码：二分的前提：有序（一提到有序，必须立马想到二分！）1. 用二分查找法，定位k的第一次出现位置和最后一次出现位置。2. 第一个判断表示是前一个不等于这个数，最后一个判断标准是后一个不等于这个数 ，若不是就继续加一或者减一
- 注意的点：

剑指offer（40）：一个整型数组里除了两个数字之外，其他的数字都出现了两次。请写程序找出这两个只出现一次的数字。

- 伪代码：这题两个思路，用哈希表和用位运算的方法。1. 位运算中异或的性质：两个相同数字异或=0，一个数和0异或还是它本身。当只有一个数出现一次时，我们把数组中所有的数，依次异或运算，最后剩下的就是落单的数，因为成对儿出现的都抵消了。

剑指offer（41）：求和为S的连续正数序列

- 伪代码：双指针技术 1. 定义两个指针，左指针从1开始，右指针从2开始 2. 求和（1+2 = 3，如果判断3小于20，右指针++，2变为3，求和3+3=6。循环一直到右指针=6，和为21。3. else if 判断21大于20，左指针++，1变为2，和减去左指针值，和为21-1=20。4. else 和与输入一样，存数，再把右指针++，求和，求剩余组合。5 循环2，3，4步直到左指针都大于S的一半

- 注意的点：

剑指offer（42）：输入一个递增排序的数组和一个数字S，在数组中查找两个数，使得他们的和正好是S。

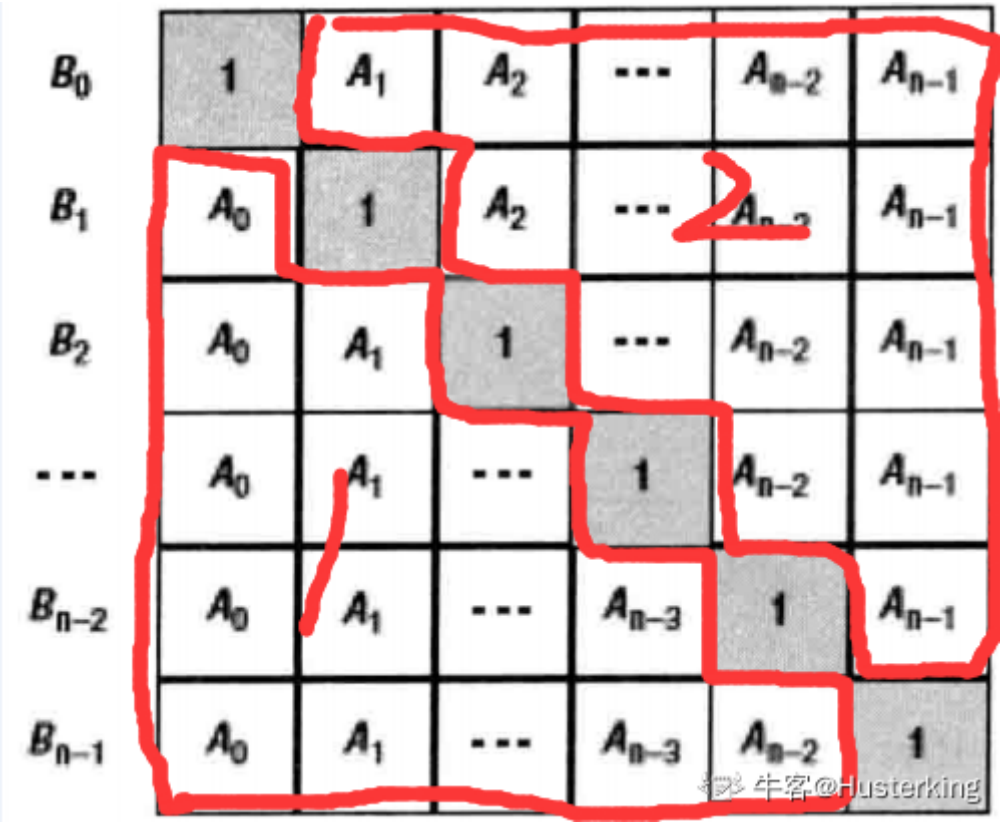
- 伪代码：1. 数列满足递增，设两个头尾两个指针i和j，2. 若ai + aj == sum，就是答案（相差越远乘积越小）3. 若ai + aj > sum，aj肯定不是答案之一（前面已得出 i 前面的数已是不可能），j -= 1 4. 若ai + aj < sum，ai肯定不是答案之一（前面已得出 j 后面的数已是不可能），i += 1 5. 循环1，2，3，4步
-

剑指offer（50）：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，请找出数组中任意一个重复的数字。

- 伪代码：哈希表法 1. 由于所有元素值是有范围的，因此可以用一个长度为n的数组，下标表示序列中的每一个值，下标对应的值表示该下标出现的次数。2. 只需扫描一次原序列，就统计出所有元素出现的次数； 3. 再扫描一次哈希数组，找到一个出现次数大于1的值即可。
- 注意的点：

剑指offer（51）：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]A[1]...A[i-1]A[i+1]...A[n-1]，即除去 A[i] 的所有值相乘。

- 伪代码：1. 可以把B[i]=A[0]A[1]....A[i-1]A[i+1]....A[n-1]。看成A[0]A[1].....A[i-1]和A[i+1].....A[n-2]A[n-1]两部分的乘积。即通过A[i]项将B[i]分为两部分的乘积。效果相当于是个对角矩阵。2. 第一个for循环用来计算上图1范围的数，第二个for循环用来计算上图2范围的数。



> 例子:

> 设有数组大小为5。
> 对于第一个for循环
> 第一步：b[0] = 1;
> 第二步：b[1] = b[0] * a[0] = a[0]
> 第三步：b[2] = b[1] * a[1] = a[0] * a[1];
> 第四步：b[3] = b[2] * a[2] = a[0] * a[1] * a[2];
> 第五步：b[4] = b[3] * a[3] = a[0] * a[1] * a[2] * a[3];
> 然后对于第二个for循环
> 第一步
> temp *= a[4] = a[4];
> b[3] = b[3] * temp = a[0] * a[1] * a[2] * a[4];
> 第二步
> temp *= a[3] = a[4] * a[3];
> b[2] = b[2] * temp = a[0] * a[1] * a[4] * a[3];
> 第三步
> temp *= a[2] = a[4] * a[3] * a[2];
> b[1] = b[1] * temp = a[0] * a[4] * a[3] * a[2];
> 第四步
> temp *= a[1] = a[4] * a[3] * a[2] * a[1];
> b[0] = b[0] * temp = a[4] * a[3] * a[2] * a[1];
> 由此可以看出从b[4]到b[0]均已经得到正确计算。

- 注意的点：

剑指offer（65）：判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。

- 题目补充路径可以从矩阵中的任意一个格子开始，每一步可以在矩阵中向左，向右，向上，向下移动一个格子。如果一条路径经过了矩阵中的某一个格子，则该路径不能再进入该格子。

$$\begin{bmatrix} a & b & c & e \\ s & f & c & s \\ a & d & e & e \end{bmatrix}$$ 矩阵中包含一条字符串"bcced"的路径，但是矩阵中不包含"abcb"路径，因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，路径不能再次进入该格子。
- 伪代码：1.

剑指offer（66）：地上有一个m行和n列的方格。一个机器人从坐标0,0的格子开始移动，每一次只能向左，右，上，下四个方向移动一格，但是不能进入行坐标和列坐标的数位之和大于k的格子。

- 题目补充：例如，当k为18时，机器人能够进入方格（35,37），因为3+5+3+7 = 18。但是，它不能进入方格（35,38），因为3+5+3+8 = 19。请问该机器人能够达到多少个格子?

*剑指offer（67）：剪绳子

- 题目补充：给你一根长度为n的绳子，请把绳子剪成整数长的m段（m、n都是整数，n>1并且m>1），每段绳子的长度记为k[0],k[1],...,k[m]。请问k[0]xk[1]x...xk[m]可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。
- 伪代码：1. 需要O(n^2)时间和O(n)空间，也就是利用一个表，储存长度为1~n绳子的最大乘积。2. 临界点其实是4，也就是说4以下的其实不论怎么分，成绩都不可能比自身的数大！那么此时存储的其实是最大的值，不一定是乘法最大值，
- 注意的点：1. 剩下的绳子长度为4时，把绳子剪成两段长度为2的绳子。 为什么选2，3为最小的子问题？因为2，3包含于各个问题中，如果再往下剪得话，乘积就会变小。 为什么选长度为3？因为当n≥5时，3(n−3)≥2(n−2)

- 动态规划求解问题的四个特征：
  ①求一个问题的最优解；
  ②整体的问题的最优解是依赖于各个子问题的最优解；
  ③小问题之间还有相互重叠的更小的子问题；
  ④从上往下分析问题，从下往上求解问题；

- 确定动态规划三要素：

  (1）问题的阶段

  (2）每个阶段的状态

  (3）从前一个阶段转化到后一个阶段之间的递推关系。

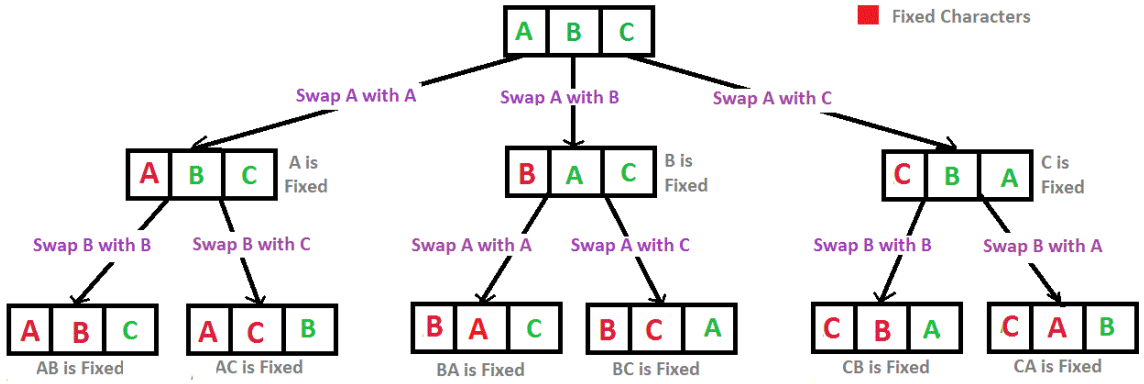
字符串

剑指offer（2）：字符串空格替换，如果有空格，就将他替换成%20

- 伪代码：1. 从前向后记录空格的数量，遍历一次　2. 从后往前插入%20，做法是由当前空格数决定决定每个字符需要后移多少格，在遇到空格时，插入%20并空格数减一
- 注意的点：这题考的是如何减少时间复杂度，在移动字符时考虑

剑指offer（27）：输入一个字符串,按字典序打印出该字符串中字符的所有排列。

- 概念：输入字符串abc,则打印出由字符a,b,c所能排列出来的所有字符串abc,acb,bac,bca,cab和cba。
- 伪代码1：回溯法的经典应用。1. 回溯法其实就是在生成棵树的过程，回溯法：其实对于回溯法，我们要从反向开始考虑。我们每次从原始数组中选择一个加入到结果中，当原始数组中（新建的）没有元素时（也就是len(a)==0，此时结果为[1,2,3]），我们得到了第一个排列，我们将这个排列加入到结果集中，然后返回上一步，也就是我们现在有[1,2]，再返回一步[1]，此时再加入3，再加入2，得到[1,3,2],



Recursion Tree for Permutations of String "ABC"

- 伪代码2：1. abc———->将第一个字符与自己本身还与其他的字符进行交换后得到abc、bac、cba。注：用一for循环即可实现。2. 在第一步每种情况的基础上，保持第一个字符不变，求剩余几位的排列。以bac为例：保持第一位不变，剩余两位ac的排列为：ac、ca（注：发现与第一步一样，因此用递归来实现）。直到保存到只剩下一个字符没有交换后加上前面的不变的字符作为输出结果，为：bac、bca。
- 注意的点：

剑指offer（34）：第一个只出现一次的字符

- 伪代码：1. 建立一个哈希表，第一次扫描的时候，统计每个字符的出现次数。第二次扫描的时候，如果该字符出现的次数为1，则返回这个字符的位置。时间复杂度为O(2n)
- 注意的点：python的哈希表建立有序字典用 collections.OrderedDict()

剑指offer（43）：左旋转字符串。字符序列S=”abcXYZdef”,要求输出循环左移3位后的结果，即“XYZdefabc”。

- 伪代码：1. 这道题考查的核心是灵活利用字符串翻转。假设字符串abcdef，n=3，设X=abc，Y=def，所以字符串可以表示成XY，如题干，问如何求得YX。假设X的翻转为XT，XT=cba，同理YT=fed，那么YX=(XTYT)T，三次翻转后可得结果。

剑指offer（44）：翻转单词顺序列

- 伪代码:1. 以空格为分界符，切分字符串可以得到一个字符串数组，对数字逆序遍历进行拼接。
- python

剑指offer（49）：输入一个字符串,包括数字字母符号,可以为空，输出如果是合法的数值表达则返回该数字，否则返回0

- 伪代码：1. 这道题要判断的边界条件非常多，数据上下的溢出、空字符串、只有正负号、有无正负号、有无非法字符，使用字典可以简化代码。2. 简单来说就是判断每个字符

剑指offer（52）：正则表达式匹配,请实现一个函数用来匹配包括'.'和'*'的正则表达式。

- 伪代码：1. 两个字符串都为空，返回true 2. 当第一个字符串不空，而第二个字符串空了，返回false，如果第一个字符串空了，第二个字符串非空，需要判断。3. pattern下一个字符为‘*’或不为‘*’，4. 不为‘*’,直接匹配当前字符。如果匹配成功，继续匹配下一个；如果匹配失败，直接返回false 5. 为‘*’时，稍微复杂一些，因为‘*’可以代表0个或多个。a>当‘*’匹配0个字符时，str当前字符不变，pattern当前字符后移两位，跳过这个‘*’符号；6. 当‘*’匹配1个或多个时，str当前字符移向下一个，pattern当前字符不变。（这里匹配1个或多个可以看成一种情况，因为：当匹配一个时，由于str移到了下一个字符，而pattern字符不变，就回到了上边的情况a；当匹配多于一个字符时，相当于从str的下一个字符继续开始匹配）

剑指offer（53）：请实现一个函数用来判断字符串是否表示数值（包括整数和小数）

- 题目：字符串"+100","5e2","-123","3.1416"和"-1E-16"都表示数值。 但是"12e","1a3.14","1.2.3","+-5"和"12e+4.3"都不是。
- 伪代码：1. 在遍历的时候判断异常情况

剑指offer（54）：字符流中第一个不重复的字符

- 题目：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，第一个只出现一次的字符是"l"。
- 伪代码：1. 创建字典，key为读取的字符串中的每一个字符，val为每个字符出现的个数的计数值 2. 输出函数：遍历字符串s中的字符，如果某个字符对应的计数为1，则返回该字符。插入函数，从字符流中读入字符到字符串s中，如果读入的字符在字符串中已存在，在字典中对应的字符计数加一，如果读入的字符在字符串中不存在，则字典中对应的字符计数为一（即新增了一个新的字符）

栈和队列

剑指offer（5）：两个栈实现队列

- 伪代码：输入数据，入栈A，输出数据从栈B弹出，当B栈空的时候，从栈A从装入栈B。
- 注意的点： 入队列直接入栈，出队列先判断是否空，空则先将栈1入栈2，不空直接出栈2
- 变形的题：**两个队列实现一个栈**

剑指offer（20）： 包含min函数的栈

- 伪代码： 双栈法 1. 一个用来存所有的元素“stackTotal”,另一个“stackLittle”用来存加入新的元素后当前stackTotal中对应的最小值。2. 两个栈中的元素数量始终保持一致，当新的元素小于“stackLittle”栈顶元素时，“stackLittle”像栈顶push新来的元素，否则，“stackLittle”向栈顶加入原栈顶元素。3. 执行“pop”方法时，两个栈同时弹出各自的栈顶元素。
- 注意的点：优化“stackLittle”的空间，当新的元素小于“stackLittle”栈顶元素时，“stackLittle”像栈顶push新来的元素，否则，“stackLittle”不变。在执行“pop”方法时，判断时候两个栈顶元素相同，相同同时弹出，不同取值不弹出“stackLittle”的值。

剑指offer（21）： 栈的压入弹出序列,第一个表示压入序列，判断第二个是不是弹出序列。

- 伪代码：模拟题 1. 新建一个栈，将数组A压入栈中，当栈顶元素等于数组B时，就将其出栈，当循环结束时，判断栈是否为空，若为空则返回true.

剑指offer（64）：给定一个数组和滑动窗口的大小，找出所有滑动窗口里数值的最大值。

- 题目补充：例如，如果输入数组{2,3,4,2,6,2,5,1}及滑动窗口的大小3，那么一共存在6个滑动窗口，他们的最大值分别为{4,4,6,6,6,5}； 针对数组{2,3,4,2,6,2,5,1}的滑动窗口有以下6个：{[2,3,4],2,6,2,5,1}， {2,[3,4,2],6,2,5,1}， {2,3,[4,2,6],2,5,1}， {2,3,4,[2,6,2],5,1}， {2,3,4,2,[6,2,5],1}， {2,3,4,2,6,[2,5,1]}。
- 伪代码：1.

树

- 建立二叉树：https://blog.csdn.net/Jingnian_destiny/article/details/87887142

剑指offer（4）：利用前序遍历和中序遍历重构二叉树

- 伪代码：1. 前序序列第一个结点确定根结点 2. 根结点在中序序列中的位置分割出左右两个子序列（对应两个子树） 3. 左右子树递归第12步求解
- 注意的点： **<递归的代码需要及时的敲>**

剑指offer（17）：输入A和B两个二叉树，判断B是A的子树。空树不是子树

- 概念：子树和子结构：1. 若 B 是 A 的**子树**，则结点值完全相同，它们俩的左子树、右子树所有结点的值也完全相同，B的根节点**必须是A的根节点的左或右的子节点** 2. 若 B 是 A 的**结构**，则结点值完全相同，它们俩的左子树、右子树所有结点的值也完全相同，B的根节点**不必须是A的根节点的左或右的子节点**
- 伪代码：遍历树结构，

剑指offer（18）：输入一个二叉树，输出一个二叉树的镜像

- 伪代码： 1. 交换两个子树，2. 递归左右子树的镜像，终止条件为当前结点为叶结点

剑指offer（22）：从上往下打印出二叉树的每个节点，同层节点从左至右打印。

- 伪代码：就是层次遍历 1. 每一次打印一个节点的时候，如果该节点有子节点，则把该节点的子节点放到一个队列的尾部。接下来到对队列的头部取出最早进入队列的节点放到list中，重复前面的操作，直至队列中所有的节点都存到list中。

剑指offer（23）： 判断数组是不是某二叉搜索树的后序遍历的结果

- 伪代码：对于一个序列S，最后一个元素是x （也就是根），如果去掉最后一个元素的序列为T，那么T满足：T可以分成两段，前一段（左子树）小于x，后一段（右子树）大于x，且这两段（子树）都是合法的后序序列。完美的递归定义 :) 。步骤1. 得到最后一个元素x 2. 左子树（序列左边部分）节点小于根节点 3. 右子树的节点（序列右边部分）都大于根节点。4. 递归判断左右子树。

剑指offer（24）： 找出二叉树中和某个整数值相等的路径

- 概念：路径为从树的根结点开始往下一直到叶结点所经过的结点形成一条路径
- 伪代码：深度优先遍历1. 实现一个深度遍历(递归调用) 2. 在主函数调用深度优先遍历，并判断路径和是否和该整数相等。

剑指offer（26）：二叉搜索树转换成一个排序的双向链表,(不能创建任何新的结点)

- 伪代码：中序遍历，1. 将左子树构造成双链表，并返回链表头节点。2. 定位至左子树双链表最后一个节点，3. 如果左子树链表不为空的话，将当前root追加到左子树链表。4. 将右子树构造成双链表，并返回链表头节点。5. 如果右子树链表不为空的话，将该链表追加到root节点之后。 6. 根据左子树链表是否为空确定返回的节点。
- 注意的点：

剑指offer（38）：输入一棵二叉树，求该树的深度

- 伪代码：1. 递归法，基础题

剑指offer（39）：判断一颗二叉树是否是平衡二叉树

- 伪代码：1. 借助一个获取树深度的递归函数，根据该结点的左右子树高度差判断是否平衡，2. 优化后的算法从下往上遍历，如果子树是平衡二叉树，则返回子树的高度；如果发现子树不是平衡二叉树，则直接停止遍历，这样至多只对每个结点访问一次。
- 注意的点：

剑指offer（57）：给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回

- 伪代码：1.二叉树为空，则返回空；2. 节点右孩子存在，则设置一个指针从该节点的右孩子出发，一直沿着指向左子结点的指针找到的叶子节点即为下一个节点；3.右孩子不存在的话，看节点是不是根节点。如果该节点是其父节点的左孩子，则返回父节点；否则继续向上遍历其父节点的父节点，重复之前的判断，返回结果

剑指offer（58）：请实现一个函数，用来判断一颗二叉树是不是对称的。

- 定义：如果一个二叉树同此二叉树的镜像是同样的，定义其为对称的。
- 伪代码：1. 首先根节点以及其左右子树，左子树的左子树和右子树的右子树相同2. 左子树的右子树和右子树的左子树相同即可，采用递归

剑指offer（59）：请实现一个函数按照之字形打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右至左的顺序打印，第三行按照从左到右的顺序打印

- 伪代码：1. nextlevel记录下层的节点数 2. tobeprint记录本层还有几个节点未打印，当其为0时，本层打印结束，重置其为nextlevel数量，而nextlevel更新为0 3. levelnums记录现在是第几层，从而判断是从左向右打印还是相反。

剑指offer（60）：从上到下按层打印二叉树，同一层结点从左至右输出。每一层输出一行。

- 伪代码：和JZ_22题一样，1. 每次出队一个元素，就将该元素的孩子节点加入队列中，直至队列中元素个数为0时，出队的顺序就是该二叉树的层次遍历结果

剑指offer（61）：请实现两个函数，分别用来序列化和反序列化二叉树

- 题目补充：二叉树的序列化是指：把一棵二叉树按照某种遍历方式的结果以某种格式保存为字符串，从而使得内存中建立起来的二叉树可以持久保存。序列化可以基于先序、中序、后序、层序的二叉树遍历方式来进行修改，序列化的结果是一个字符串，序列化时通过 某种符号表示空节点（#），以！ 表示一个结点值的结束（value!）。

  二叉树的反序列化是指：根据某种遍历顺序得到的序列化字符串结果str，重构二叉树。
- 伪代码：1. 递归遍历二叉树的节点，空节点使用#代替，节点之间使用逗号隔开，返回字符串 2. 设置序号index，将字符串根据逗号分割为数组，根据index的值来设置树节点的val，如果节点的值为#，则返回空的树节点。

剑指offer（62）：给定一棵二叉搜索树，请找出其中的第k小的结点。

- 题目补充：例如， （5，3，7，2，4，6，8） 中，按结点数值大小顺序第三小结点的值为4。
- 伪代码：1. 二叉搜索树的中序遍历就是树节点值的递增排列！

剑指offer（63）：如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。

- 伪代码：1. 因为要求的是中位数，那么这两个堆，大顶堆用来存较小的数，从大到小排列，小顶堆存较大的数，从小到大的顺序排序，显然中位数就是大顶堆的根节点与小顶堆的根节点和的平均数。2.

数学

剑指offer（7）：输入一个整数n，输出斐波那契数列第n项：

- 伪代码：1. 用sum存储第N项，one储存第N-1项，two储存第N-2项 2. 循环，并交替储存第N，N-1,N-2项的数，sum=one+two得到结果。
- 注意的点：递归时间复杂度是[O(N的平方)]，上面的伪代码是O(N)。

剑指offer（8）：青蛙跳台阶，青蛙可以挑1阶，也可以2阶，跳到n阶，几种做法

- 伪代码：1. 从n-1到n跳一下，剩下n-1有f(n-1)种跳法，从n-2到n跳一下,剩下n-2有f(n-2)种跳法 ,总的就是f=f(n-1)+f(n-2) 2.可以转化为斐波那契数列
- 注意的点

剑指offer（9）：青蛙变态跳，可以跳1到n阶，随便跳。求n阶几种方法

- 伪代码： 找规律，1. f(1) = 1，f(2) = f(2-1) + f(2-2) ，f(3) = f(3-1) + f(3-2) + f(3-3) ……f(n) = f(n-1) + f(n-2) + f(n-3) + … + f(n-(n-1)) + f(n-n) 2. 由： f(n) = f(n-1)+f(n-2)+…+f(n-(n-1)) + f(n-n) =f(0) + f(1) + f(2) + f(3) + … + f(n-1)和f(n-1) = f(0) + f(1)+f(2)+f(3) + … + f((n-1)-1) = f(0) + f(1) + f(2) + f(3) + … + f(n-2)得到规律： f(n) = 2*f(n-1) 3. 至此问题转化为类似斐波那契数列

剑指offer（10）：矩阵覆盖，用2*1的小矩形横着或者竖着去覆盖更大的矩形，求n个2*1有多少种方法覆盖2*n矩阵

- 伪代码： 同样找规律 1. n = 1时 f=1；n = 2时，f=2；n = 3时，f=3 ，当n=n时，分两种，当第n横着放，则还剩n=n-1种方法，若第n竖着放，则还剩n=n-2种方法。2. 由上面的规律发现本质同样还时斐波那契数列，用斐波那契额数列求解法求解。

剑指offer（12）：数值的整数次方，double类型的浮点数base和int类型的整数exponent，求base的exponent次方

这题有多种解法：

- 伪代码1：简单递推法：1. 结果设为1.0，即当exponent=0 的时候，就是这个结果2. 获取指数的绝对值 3. 根据指数大小，循环累乘（N次循环） 4. 根据指数正负，返回结果
- 伪代码2：快速幂算法：1.

剑指offer（31）：整数中1出现的次数（从1到n整数中1出现的次数）

- 伪代码：1. 思路是分别计算个位、十位、百位……..上出现 1 的个数。 以 n =216为例： 3. 个位上： 1 ，11，21，31，…..211。个位上共出现（216/10）+ 1个 1 。因为除法取整，210~216间个位上的1取不到（n=211怎么办，这里把最后取到的个位数为1的单独考虑） 4. 十位上：10~19，110~119，210~216. 十位上可看成 求（216/10）=21 个位上的1的个数然后乘10。这里再次把最后取到的十位数为1的单独拿出来，即210~216要单独考虑 ，个数为（216%10）+1 .这里加8就避免了判断的过程。 （a+8的巧妙之处在于当a的最后一位(当前分析位)为0或1时，加8不产生进位，这是为需要单独算的特殊情况做准备，而当前分析位为2~9时，不需要考虑特殊情况，所以允许加8产生的进位） 后面以此类推。
- 注意的点：

剑指offer（33）：求按从小到大的顺序的第N个丑数。

- 概念：把能够分解质因子2、3和5组合成的数称作丑数（Ugly Number）。例如6、8都是丑数，但14不是，因为它包含质因子7。
- 伪代码：丑数能够分解成2^x * 3^y * 5^z,只需要把得到的丑数不断地乘以2、3、5之后并放入他们应该放置的位置即可

- 注意的点：难点就在于如何有序的放在合适的位置。 1乘以 （2、3、5）=2、3、5；2乘以（2、3、5）=4、6、10；3乘以（2、3、5）=6,9,15；5乘以（2、3、5）=10、15、25；从这里我们可以看到如果不加策略地添加丑数是会有重复并且无序。2*x*，3y，5*z中，如果x=y=z那么最小丑数一定是乘以2的，但关键是有可能存在x》y》z的情况，所以我们要维持三个指针来记录当前乘以2、乘以3、乘以5的最小值，然后当其被选为新的最小值后，要把相应的指针+1；因为这个指针会逐渐遍历整个数组，因此最终数组中的每一个值都会被乘以2、乘以3、乘以5，也就是实现了我们最开始的想法，只不过不是同时成乘以2、3*、5，而是在需要的时候乘以*2*、3、5.

剑指offer（47）：求1+2+3+...+n，要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句（A?B:C）。

- 伪代码：1. 需利用逻辑与的短路特性实现递归终止 2. 当n==`0`时，(n>`0`)&&((sum+=Sum_Solution(n-`1`))>`0`)只执行前面的判断，为`false`，然后直接返回`0`； 3. 当n>`0`时，执行sum+=Sum_Solution(n-`1`)，实现递归计算Sum_Solution(n)。

剑指offer（48）：不用加减乘除做加法

- 伪代码：1. 相加各位的值，不算进位，二进制每位相加就相当于各位做异或操作；2. 计算进位值，相当于各位做与操作，再向左移一位。3. 重复上述两步， 各位相加 ， 计算进位值。进位值为0，跳出循环。
- 注意的点：python 需要来 & `0xFFFFFFFF` # 考虑负数

二进制(位运算)

剑指offer（11）：二进制中1的个数

- 伪代码：1. 思想一个整数不为0，那么这个整数至少有一位是1。如果我们把这个整数减1，那么原来处在整数最右边的1就会变为0，原来在1后面的所有的0都会变成1(如果最右边的1后面还有0的话)。其余所有位将不会受到影响。2. 用减1之后的数和原来的数相与运算，得到的数为原来的数最右的1变为0的数(1的个数减1)， 3. 循环第二步，得到结果
- 需要注意： 在python中进行位运算之前要判断正负，如果是负数需要n & 0xffffffff

其他，模拟题

剑指offer（45）：扑克牌顺子

- 伪代码：1.进行排序 2.计算0的个数 3.如果存在相等元素，不是顺子 4.如果缺失元素个数大于0的个数，不是顺子，反之就是顺子（也就是计算最大元素和最小元素的差，需要小于0的个数加1，再加上非0元素的个数）

剑指offer（46）：圆圈中最后剩下的数

- 伪代码：1. 用环形链表模拟圆圈的经典解法。