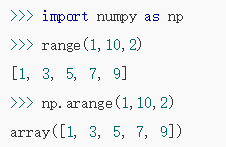
**1. Two sum问题（题号1，easy）**

**（1）range & arange**

range(start, end, step)，返回一个list对象，起始值为start，终止值为end，但不含终止值，步长为step。只能创建int型list。

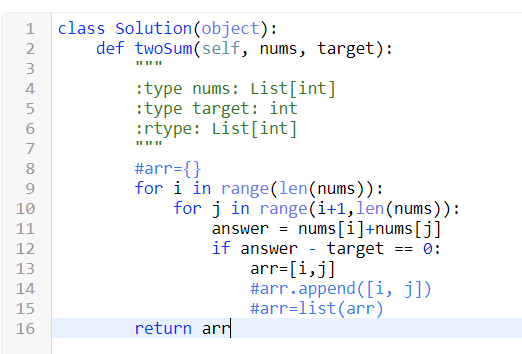
arange(start, end, step)，与range()类似，但是返回一个array对象。需要引入import numpy as np，并且arange可以使用float型数据。



**（2）我的方案**

思路：用每一个数字和其后边所有数字以此相加，找到目标值，返回此时的索引号

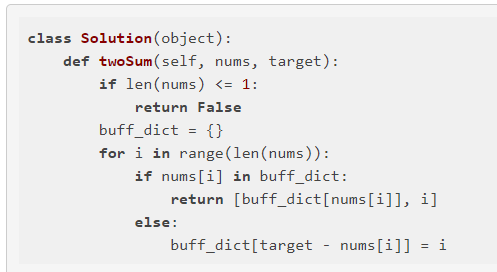
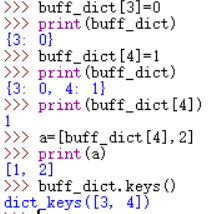
考虑两点：一是不能和自身相加，因此j+1，二是可能有多种结果，所以append;



遗留问题：若用append结果是[[]]形式，怎么把二维数组转一维，但若不考虑重复，可以按上述来写；

运行时间：32ms

**（3）题中方案：**

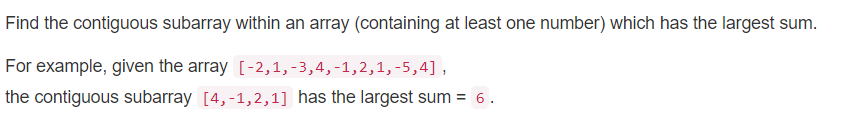
时间：26ms

Nums=[3,2,4], target = 6;

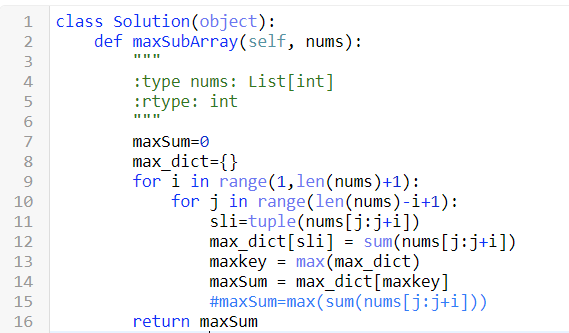
理解：在一开始啥都没有时候创建一个字典，就是说我第一个数和目标差3，那么我就把差的3和我本来3的索引（0）做成一个字典，当后边出现3时，字典就可以查到这个3，然后告诉他前边有个索引是0的3在等它，然后返回这两个3的索引；以此类推，2进来时，告诉他他和目标差4，那么当后边遇到4时，就告诉他前边有个2在等它，把他俩的坐标返回即可，这种方法的好处是不用逐个尝试，前边有记录谁缺什么

**2. Maximum subarray问题（题号53，easy**）

问题：求连续的最大和：

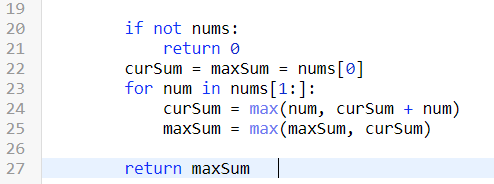


1. 初始思路：创建一个字典，使连续子列表作为键，其和作为值，max这个字典，即可得到其最大值对应的键，但是行不通，因为列表是不可哈希的，因此不可以作为键，大多数的python对象都可以为键，但是列表和字典都是不可哈希的，不能为键，因此可以改为元组



错误：理解问题：输出最大值即可，不需要子列表，因此14行又取了一遍字典里的值，但是结果仍有问题，此时结果为后六个数之和5，i,j有问题

1. 题中方案：

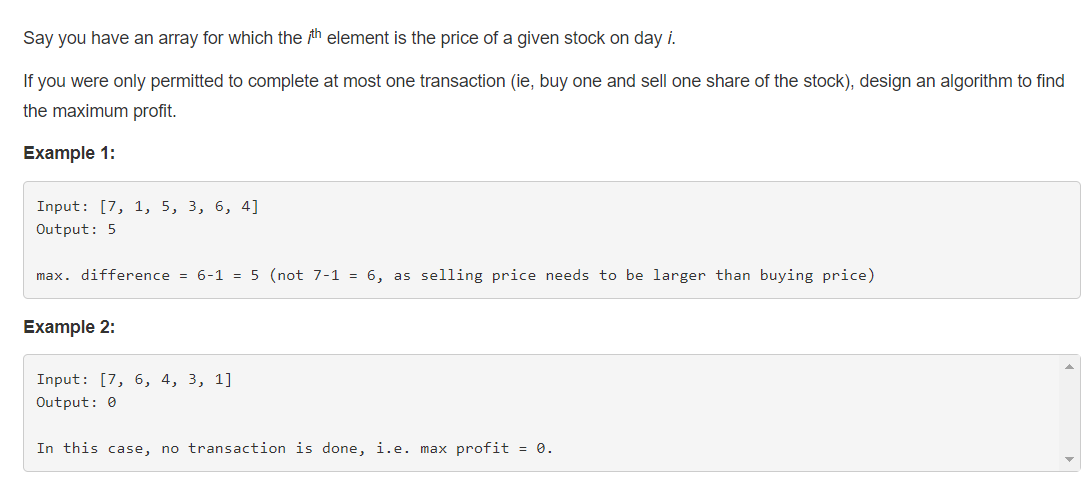


这个思路就是相当于把前两个数先相加，结果与第二个数比，若比它小，则相当于拖后腿，放弃第一个数，从第二个数开始往后加，若比第三个数大，则保留这两个的结果，接着往后加第四个数，结果若比第四个数小则放弃前三个，从第四个数开始加，也就是说从前往后依次相加，每次加两个数，把和与后一个数相比，若大则留作为新的起点，若小则弃，并使后一个数变成新的起点，再将新的数与后一个相加，同理，同时把新的数与和的最大值保存下来，以确定到哪里停止，直至加到最后一个数。

这个方法的优点也在于不用逐个相加，而是先分析数据间的关系，始终排除前边拖后腿的数，并在收集过程中的最值，以便取出，不能不取最值一直加到最后，那样和值有可能变小。

**3. Best Time to Buy and Sell Stock问题（题号121，easy）**

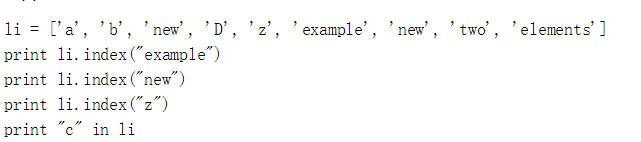
题目：给定一组股票的价格，只有一次交易（一次买入和卖出），设法使获利最大，注意买入一定在卖出之前

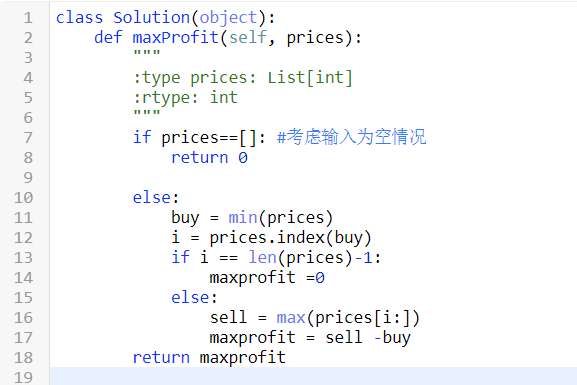


1. 我的方案：

思路：先找最小值，然后返回他的索引，然后找它之后的最大值，二者相减，若之后无最大值（或者说无值），返回为0，

返回数组的索引号用的是index

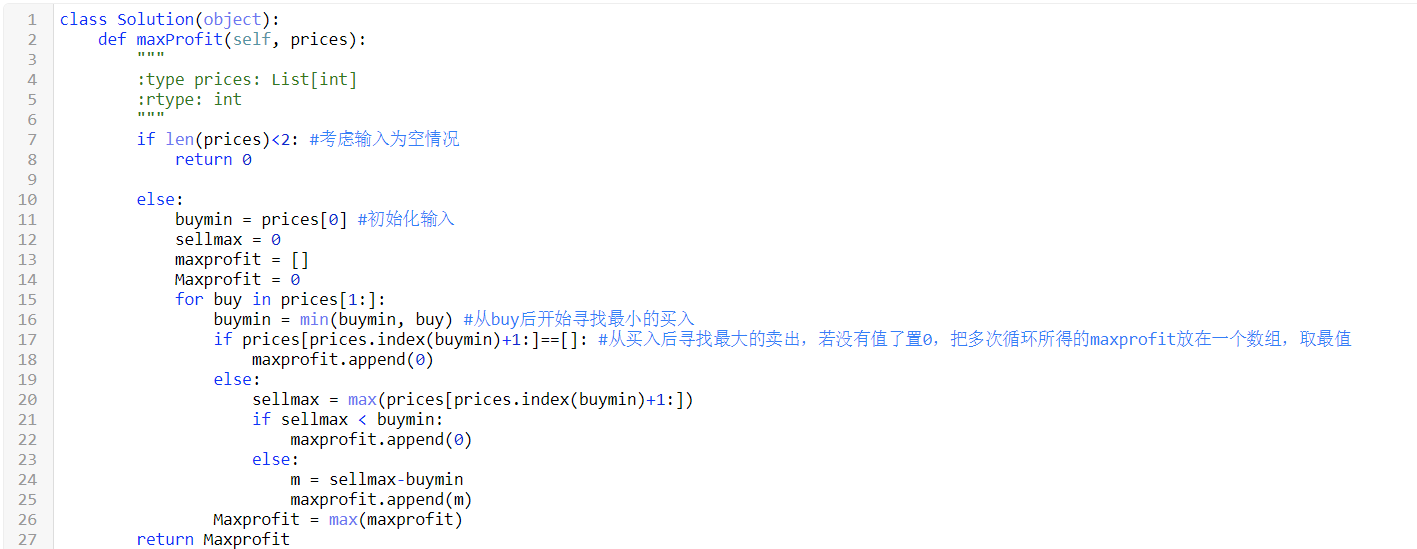




这个思路**有误**，因为不应该找最小值，而是只要买了之后有更高的价格，均可以卖出。如输入[2,4,1]，最小值1后边没有值了，但是可以2买入，4卖出

**因此需要改**：

思路：在循环内，先确定一个买入的最小值，然后从其后找最大值，若没有值了返回为0，若有最大值则计算最大利润 ；在循环外，往后继续找最小值，若找到新的最小值，再从其后找最大值，将循环中所得的最大值不断补充到利润数组里，最后取出数组中的最大值即可

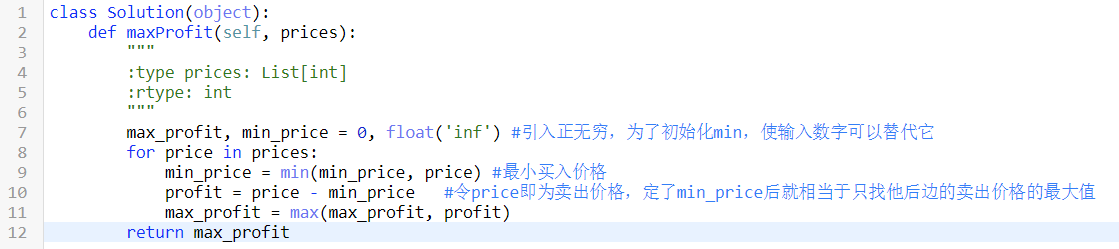


问题：超时26ms

1. 更改版：

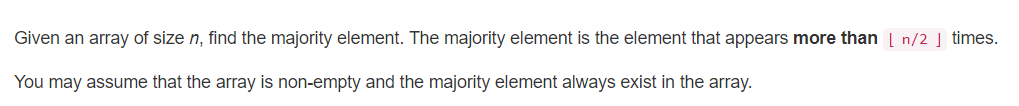
思路：前一部分与我思路一样，先给定一个最小值，但是巧妙之处在于他用了一个正无穷也就是inf来初始化最小值，这样第一个输入很容易就把它代替了，然后用prices数组的每一个数以此作为卖出价格，这样的好处在于不用考虑初始数据是不是空，或者是不是只有一个，买入和卖出的初值都是有的，并且如这种情况相减能保证为0；之后在循环中用price来做卖出值，然后及时取最大更新利润，就不用利润数组的出现了

与我的主要区别在于第10行的加入，用price当卖出，当最小买入定了的时候相当于用price卖出来以此与之作差，因此相当于自动把握了时间顺序去其后找卖出，同时还会更新新的最小值，使步骤简化了许多，值得学习。

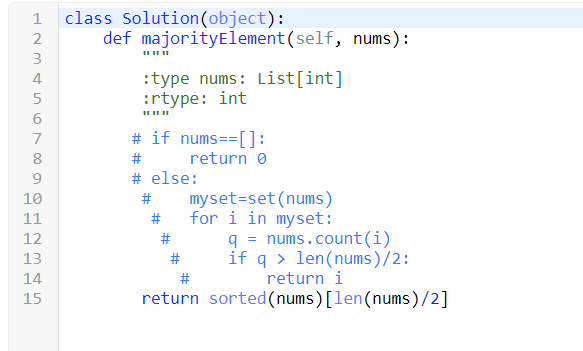


**4. Majority Element（题号169，easy）**

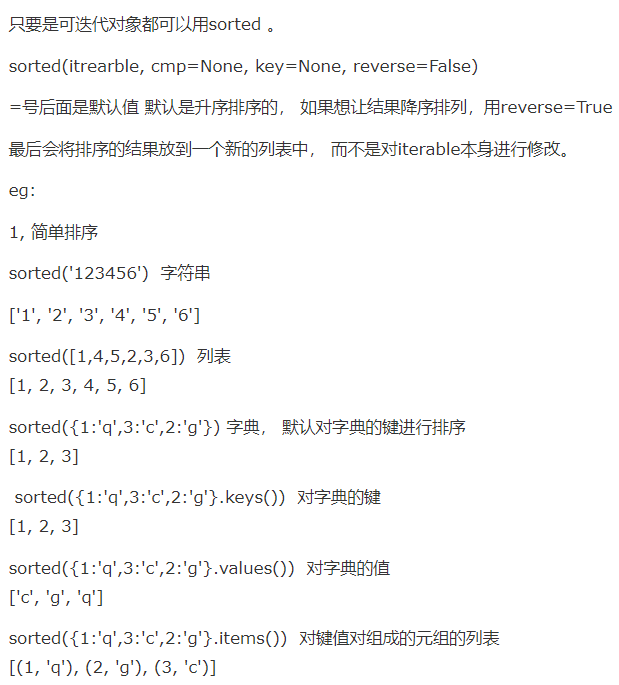
找出数组中出现数目大于一半的元素

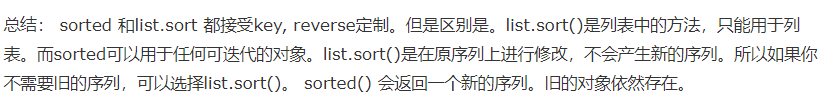


1. 我的思路：用set把非重复元素找出来，再用count逐个计数，选出大于半数的元素
2. 别人思路：用sorted把数组排序，输出其中位数（排序后的中位数一定过半了）

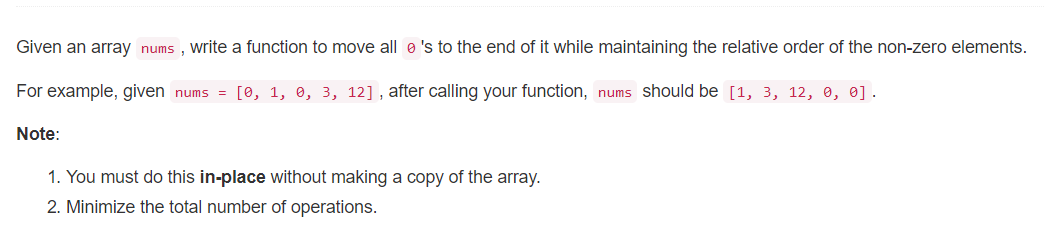


**Sorted()与list.sort()的用法**



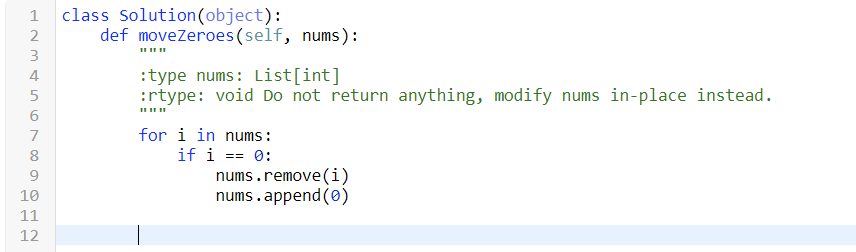


**5. Move Zeros（题号283，easy）**



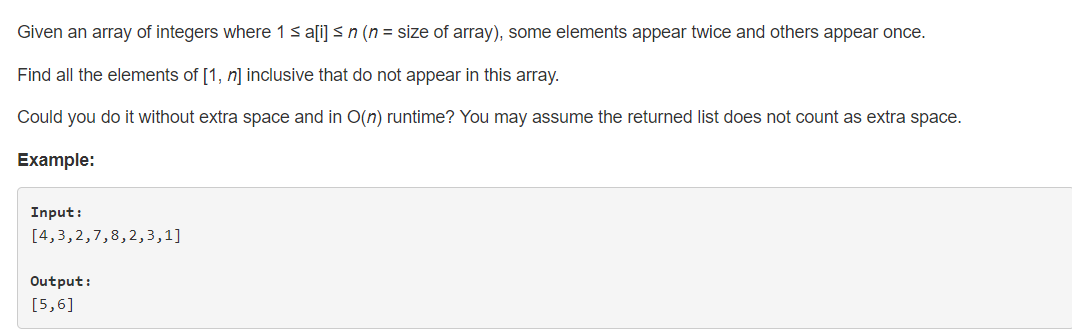
题目：把数组中的0移到后边，非零元素顺序保持不变

思路：逐个检测元素，是0删掉，后边增加0，非零不管

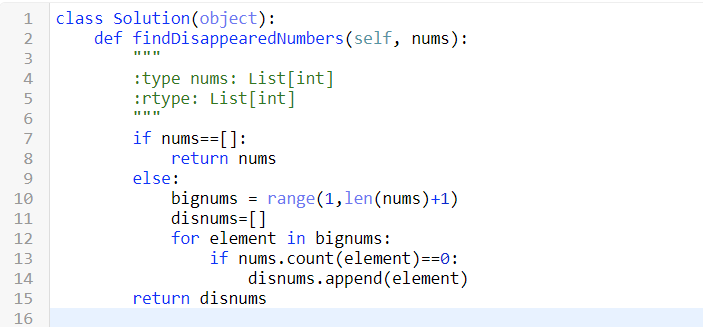


**6. Find All Numbers Disappeared in an Array（题号448，easy）**

题目：找到数组中消失的元素，注意n是数组长度



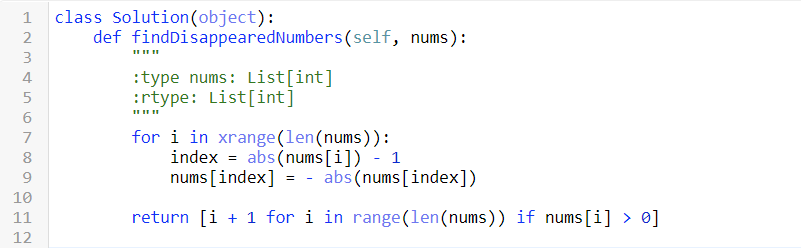
思路：构建一个大的数组，即1到n的，把原数组当做小的数组，然后在小的数组中计数大数组的每一个元素，要是计数为0，则说明确实，把这个放在缺失元素的数组里



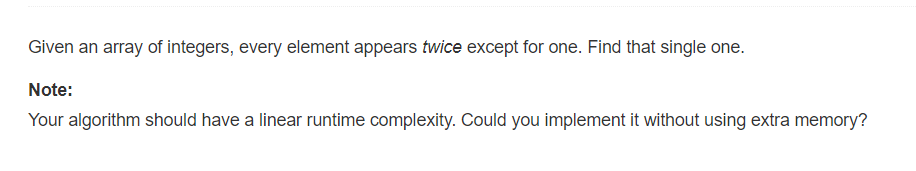
问题：超时

解决的方案：

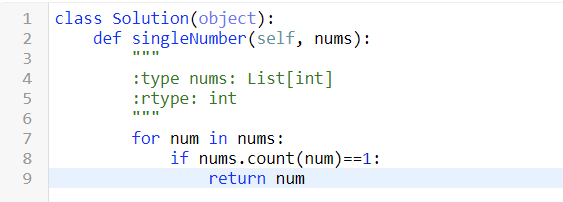
第一次遍历：用相反数的形式标记出现过的数字（下标）；第二次遍历：找出没有被标记的数字（下标），并返回



**7. 找出单个数字（题号136，easy）**



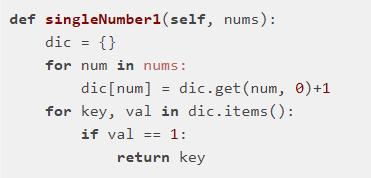
我的思路：遍历数组，计数每个元素的个数，若为1则输出该元素



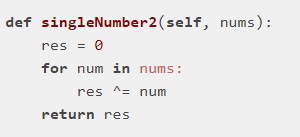
问题：超时

解决方案：

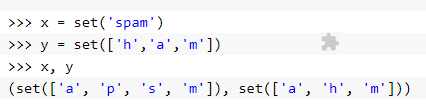
法一：建立字典，把个数作为值，用get函数，get函数返回字典中要查找的值，若不存在时返回默认值



法二：按位异或



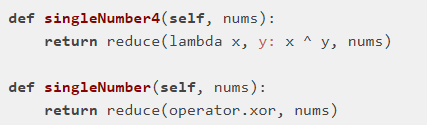
法三：用set删除数组中的重复元素



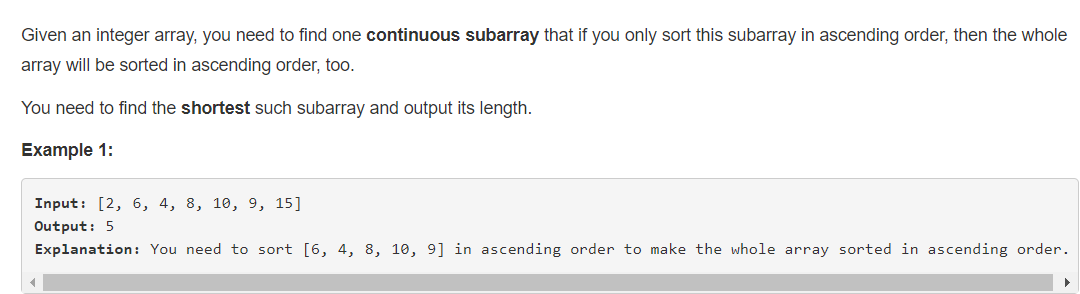


法四：reduce函数：





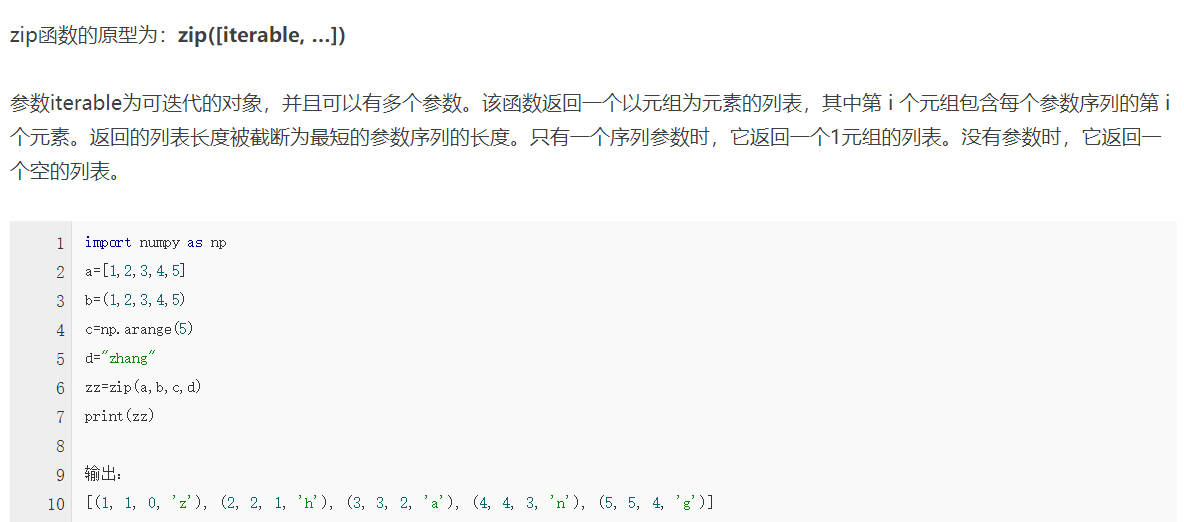
**8. 最短乱序子集（题号581，easy）**

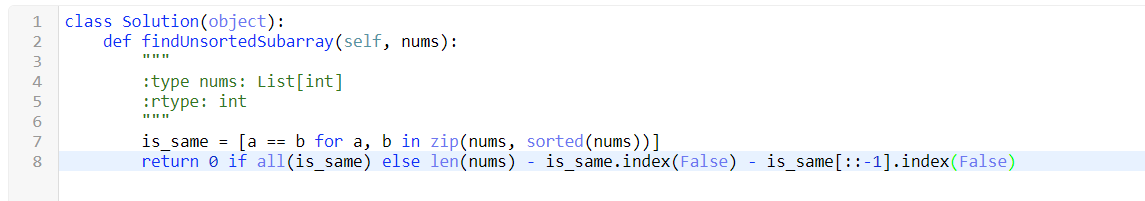


题目：找出数组中乱序的最短子集

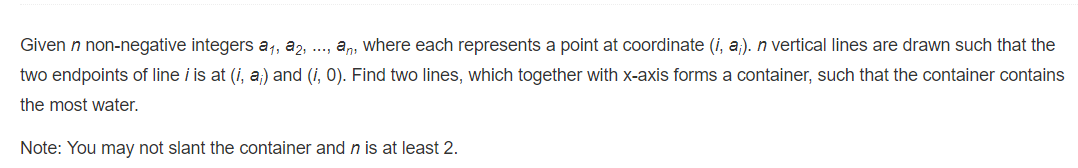
参考方案：

把乱序和正序的每个元素组成一个元素对（zip），判断它俩是否相等（a==b），相等返回True，不等返回False，最短乱序的子集就是在第一个和倒数第一个（[::-1]）之间的元素，因此用总长度减去前后的长度，即可得中间乱序的长度



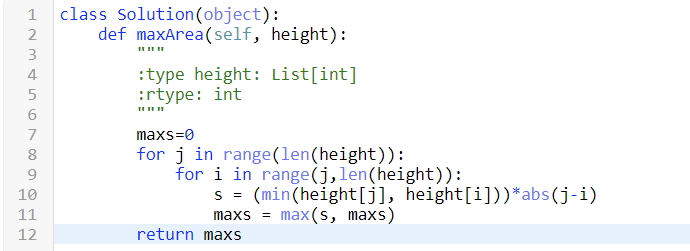


**9. 容器含水最多（题号11，medium）**



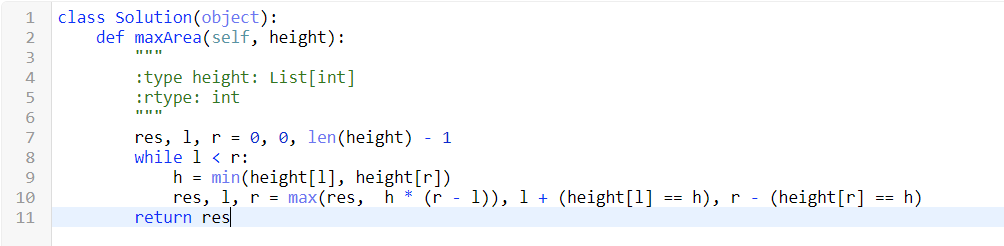
数组元素与标号组成对画在坐标系里，任取两根垂直与x轴的线，求与x轴围成的最大面积

我的思路：定一根线，变另一根线，求面积，找最值



问题：超时；

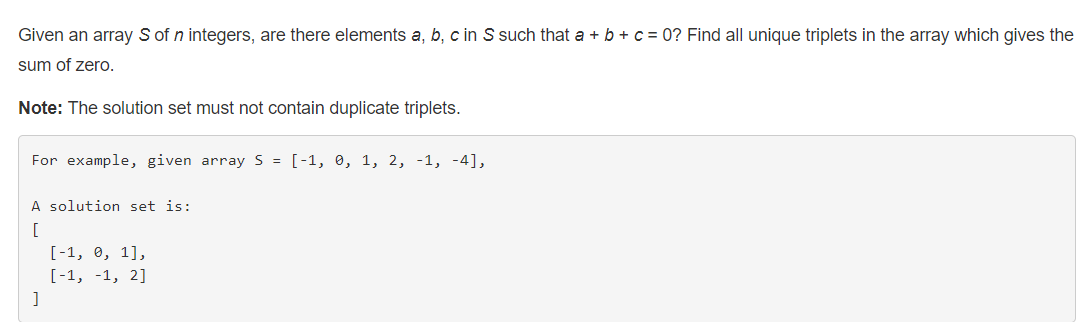
解决方案：



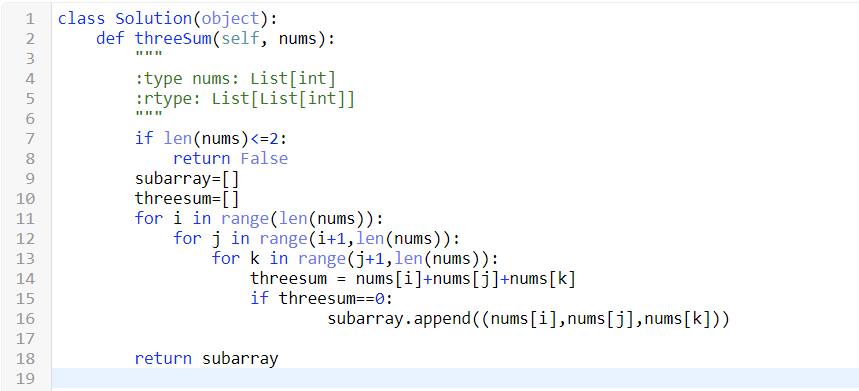
思路与我一致，但是亮点在于，既然是面积那么线的前后顺序无关，也就是说i和j与j和i表示了同一块面积，因此可以把l 和r一个从前往后，一个从后往前更新，只要两个没走岔了，就都有面积，然后找出最大即可，时间省了一半

**10. 3Sum（题号15，medium）**

题目：找出数组中求和为0的三个元素，并且要求加和情况唯一，即每个子集合元素不重复

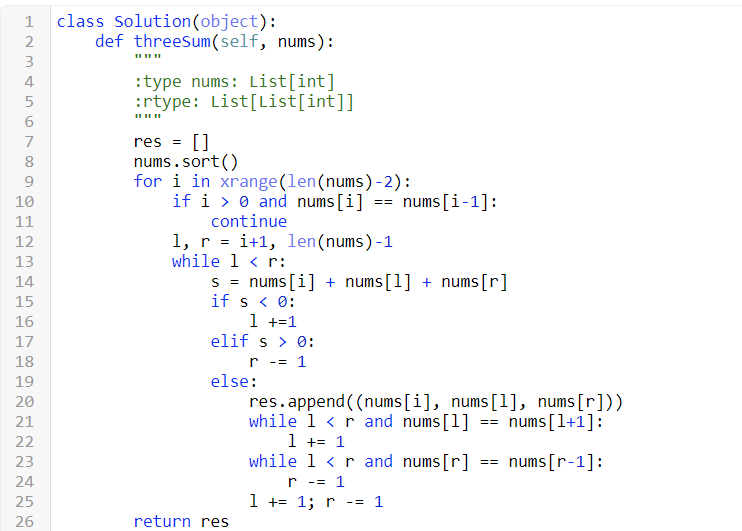


我的思路：所有三个元素加和，找出为0的情况，但是并没有解决元素唯一问题



解决方案：

为了简化运算，先给数组排序，这样前边都是小于0的，后边是大于0的，前两个数从前边开始，最后一个数从后往前，直到第二个数与第三个数相遇为止。巧妙之处有二，一是排序后若加和大于零，则往小调节第三个数，若小于0则往大调节第二个数，不用盲目求和；二是解决重复问题，因为排序，若有相同的就在一起了，因此第二个数与第三个数移动的过程中，遇到重复的跳过即可，就能够避免相同的情况

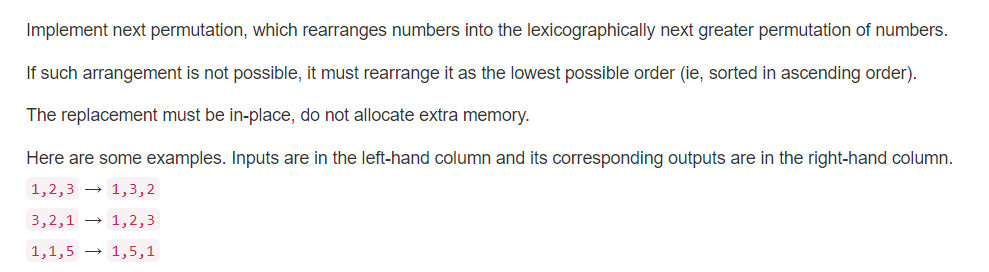


里边xrange需要说明：



**11. Next Permutation（题号31，medium）**

题目：求下一个排列比当前更大的排列，若没有这样的排列，则给它升序排列



题意理解：eg:123-132-213-231-312-321(将123进行排列组合，保证下一个比前一个大，“下一个函数”需要输入排列组合，输出下一个)，我理解的是只排列排序时的第一步

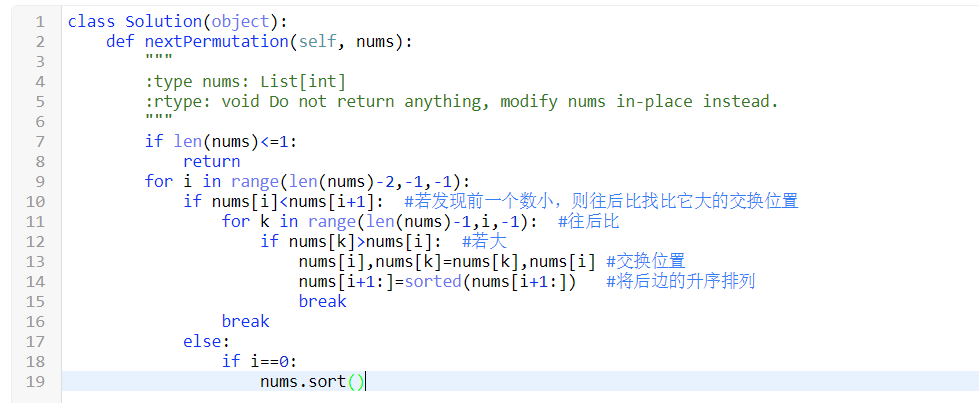
思路：

（1）由于下一个数比上一个数大，因此需要从后往前扫描，找到递增的位置设为元素i,j，(i小于j)

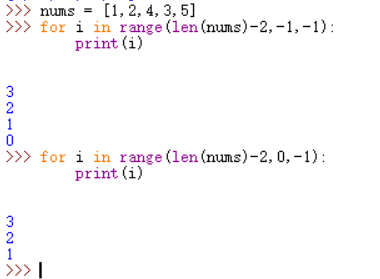
（2）由于下一个数虽然大于上一个数，且最接近上一个数，因此找到元素i，在i元素后面找到最接近i且大于i的元素k。由于i后面的元素都是降序排列的，只需要从后往前扫描找到第一个比i大的元素即可

（3）找到将i和k交换位置，然后将k后面的元素递增排序

（4）找不到，则将全部元素递增排序

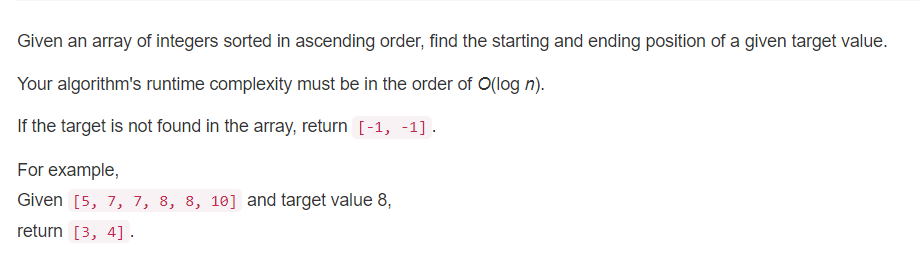


再补充一点range的用法，它有三个参数，第一个是起点，第二个是终点，第三个是步长，相当于起点一步一步加步长，最后到终点为止得到这么一列数，如果倒着取，那个起点就是原终点，终点是原起点（但这时原起点不会被包含，所以取-1，相当于原起点的左边），步长是-1

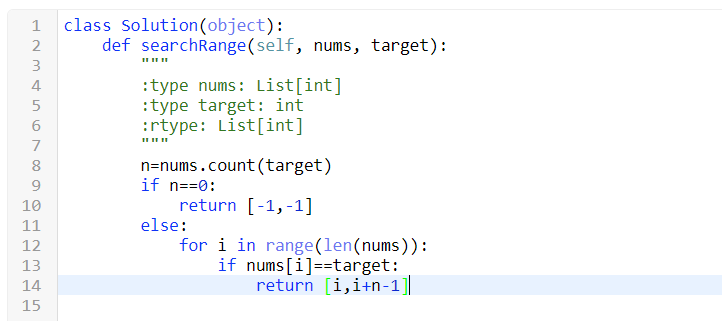


**12. Search for a Range（题号34，medium）**

题目：在一个升序排列的数组中，寻找目标元素的起点和终点索引，若没有这个元素，返回[-1,-1]

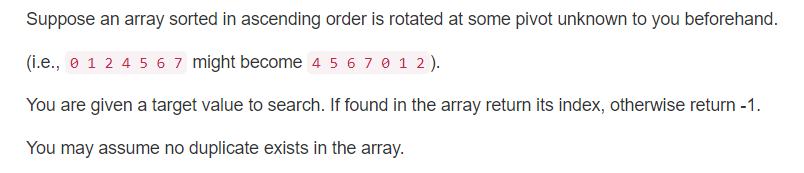


思路：先计数目标元素个数，若为0直接返回[-1,-1]，若不为0，则寻找其起点，然后加上n-1则为终点



**13. Search in Rotated Sorted Array（题号33，medium）**

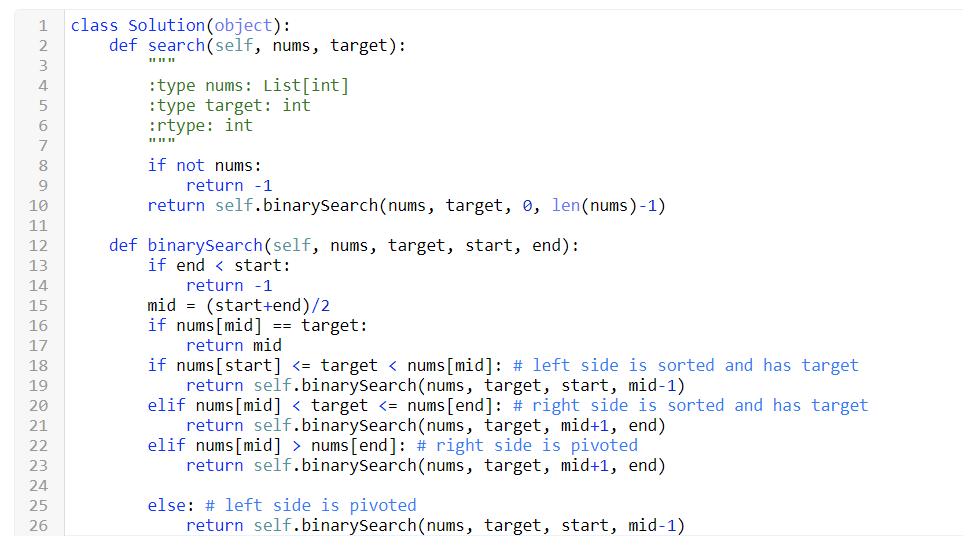
题目：在一个部分旋转的数组中找到目标元素



理解意图：由于数组部分旋转，因此不能用一般的二分法查找，但是每一部分是升序排列的，因此小部分内可以用二分法，这个题的亮点在于二分法的迭代以及分情况讨论：

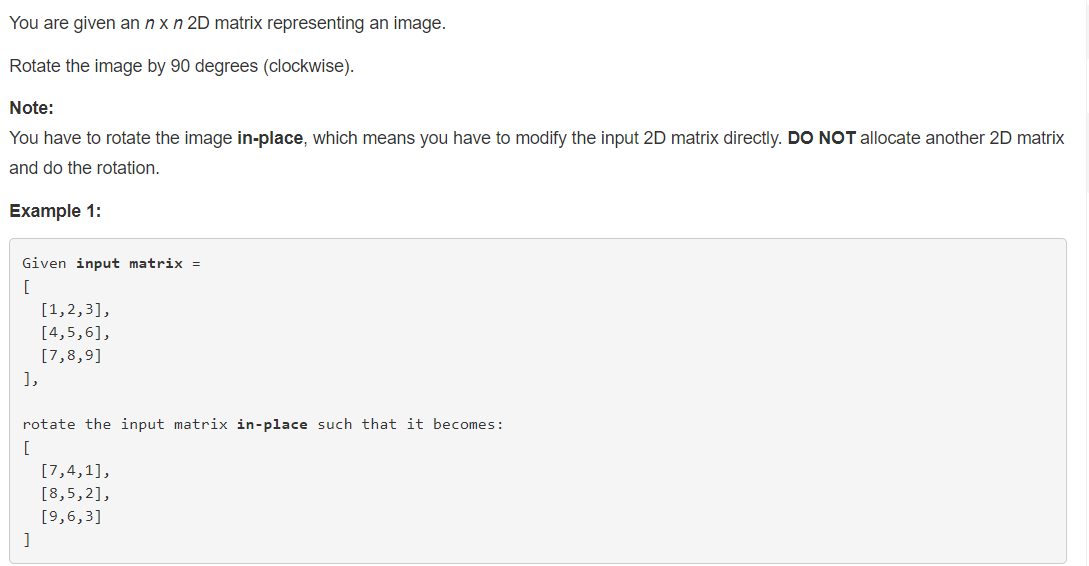
输入三个元素：起点，终点以及目标，然后求终点，如果：

1. 目标在起点与中点之间，则在左半部分继续找；
2. 目标在中点与终点之间，则在右半部分继续找；
3. 若都不符合两者情况，那么当中点大于终点时，右侧旋转了，并且由于不符合情况（1），因此往右接着找；
4. 若不符合上述三种，则左侧旋转了，往左接着找



**14. Rotate Image（题号48，medium）**

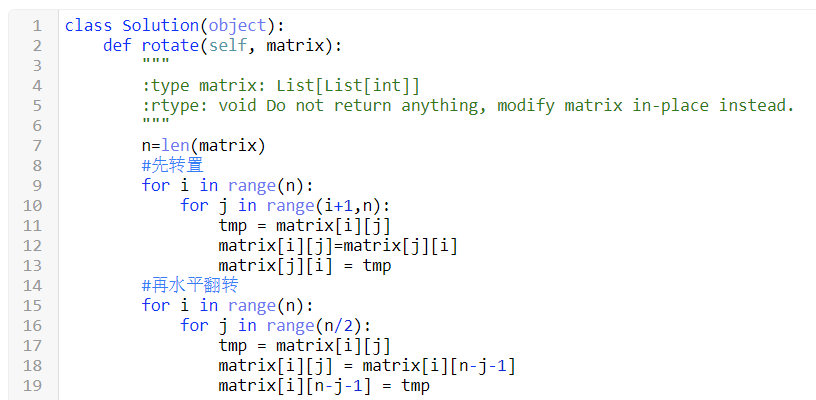
题目：将矩阵顺时针旋转90度



思路：先转置，在水平翻转

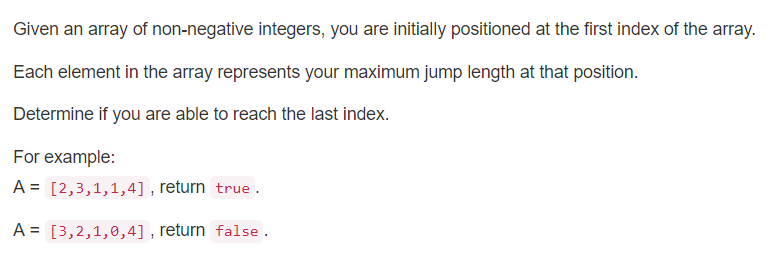
1. 转置就是两个维度索引对调
2. 水平翻转就是第一个维度（行，i）不变，第二个维度两个对称位置的元素对调

求对称的位置时，j最大到n/2，因为相当于只有一半的索引在变换，总长减去一半的即可，注意n-j-1的使用



**15.Jump Game（题号55，medium）**

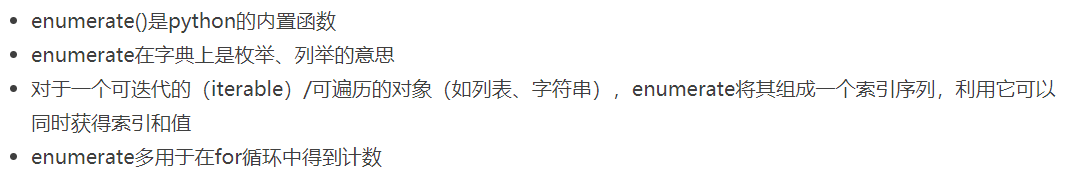
题目：给定一个整型数组，你在第一个位置，每次可以最多跳当前元素所代表的步数（可以跳的小于这个数），若能够到达最后一个元素，则返回真，到不了则返回假



解决思路：求出每个位置能到达的最远位置，若没有位置能到最后一个元素则返回为假

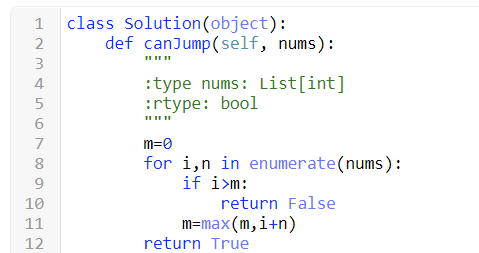
误区：当前元素是能跳的最大步数，可以跳的小于它，但不可以超过它，也不一定非得是这个步数

代码：enumerate是一个可以同时返回索引和元素值的函数，具体用法如下：



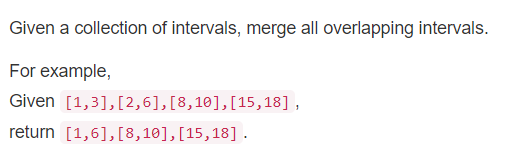


因此我们这时索引值加上元素值，是它在当前位置能够到达的最远的位置，在遍历所有元素时不断更新这个最大值，当这个最大值小于某一索引时，就代表没有步数可以到达当前的位置，也就是无法到达最后一个元素

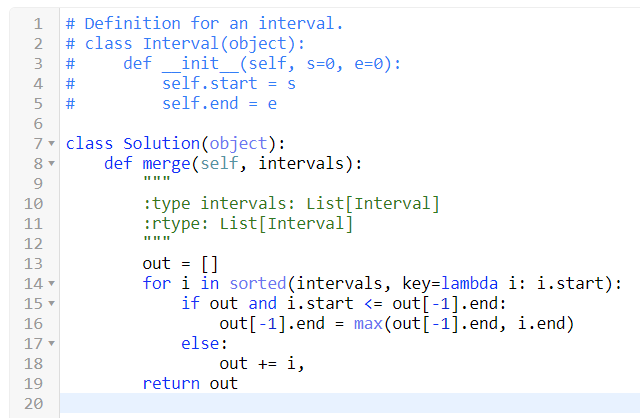


16. Merge Intervals（题号56，medium）

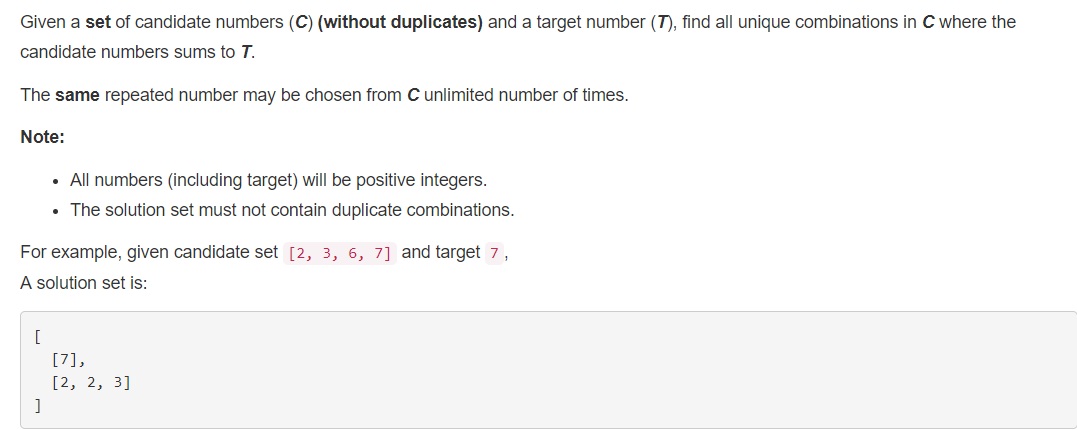
题目：给定一组时间间隔，把所有重叠的间隔合并



答案解决办法：只需通过开始坐标进行排序，或者将当前的时间间隔与之前的时间间隔结合起来，或者如果它们不重叠，就把它添加到输出中。

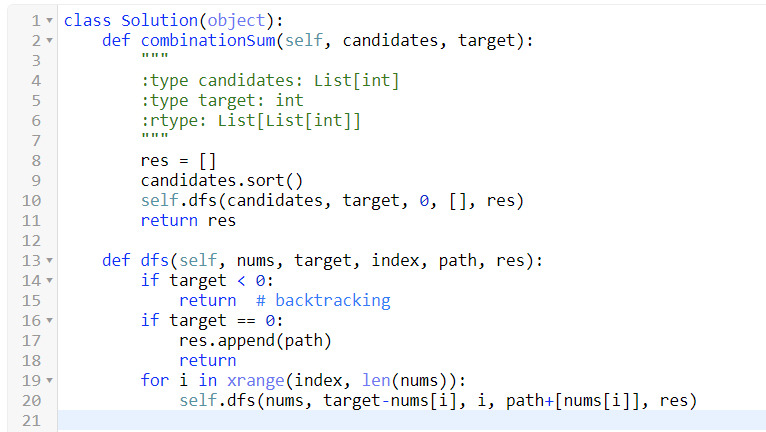


**17.Combination Sum（题号38，medium）**



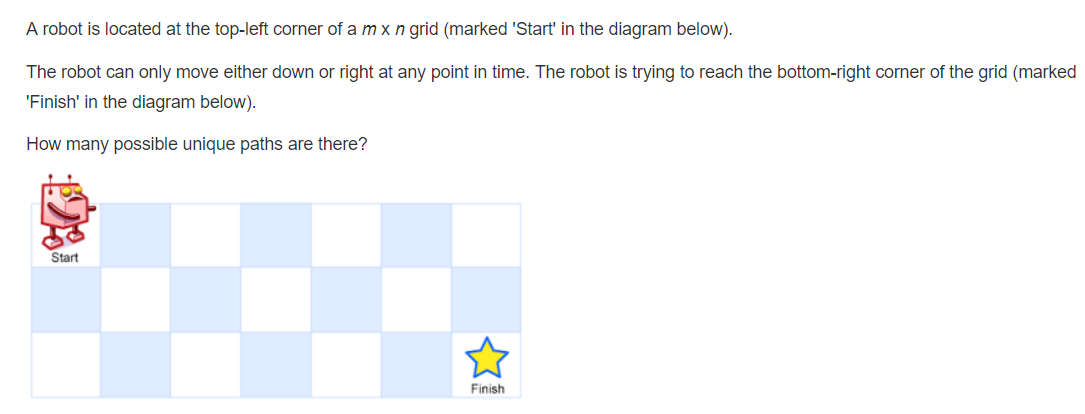
题目：给定一个目标，列出数组中所有相加等于他的组合，每个元素使用次数不限

思路：穷举递归法



**18.Unique Paths（题号62，medium）**

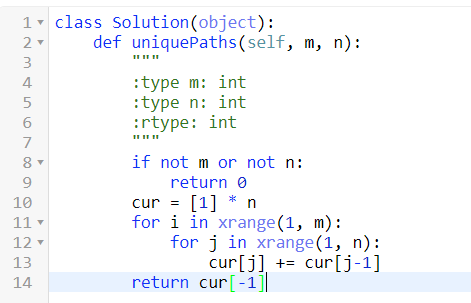
题目：只能向右和向下，求起点到终点的路径数量



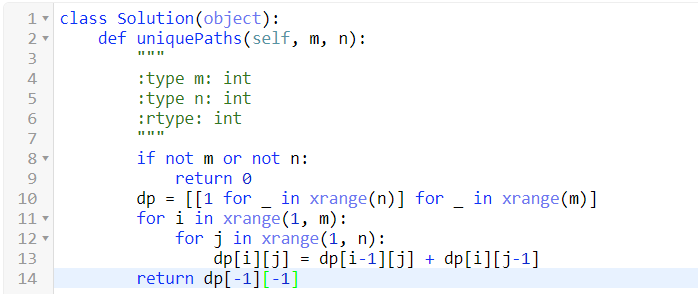
思路一：（一种时间复杂度为O（n）的简单解法）

总体上想：最后一个格到达的次数就是路径的总方法数，因此最后返回的是最后一格的次数；小块想：把每四个格看做一个整体，到达最右下角的次数就等于它左边的和上边的次数加和，若只有四个格，那么走左边和走右边各一次，因此是2，当格数往下和右扩的时候，分别求出新格左侧和上侧的值然后加和即可。

操作：开辟一块初始值为1的n维行向量，每当j递增的时候，相当于向右移动的次数增加，而i递增时相当于向下的次数增加，因此通过循环自带m\*n的效果，因此只用n维就可以，大循环向下走，小循环向右走，很巧妙。

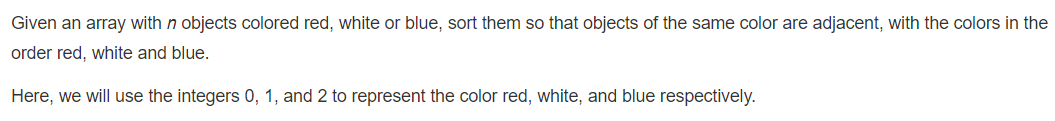


思路二：就是开辟一个m\*n的地盘，然后用二维的计算，还是用左边和上边的相加



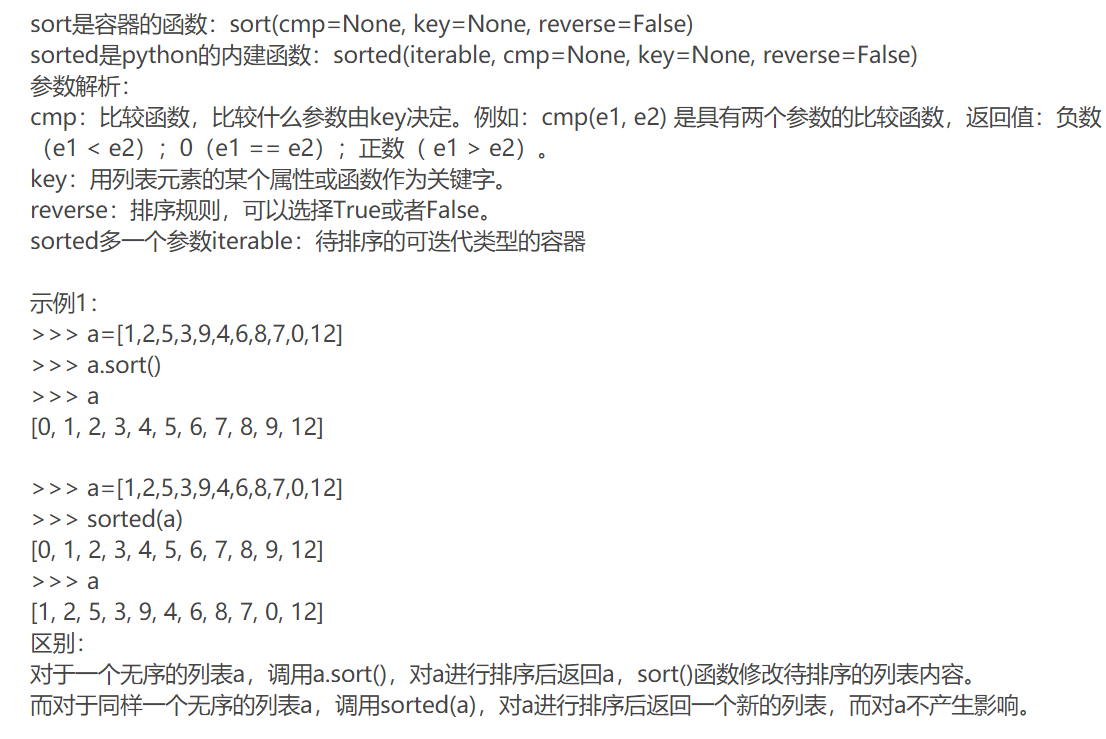
**19.Sort Colors（题号：75，medium）**

题目：给颜色分类，相同颜色放一起



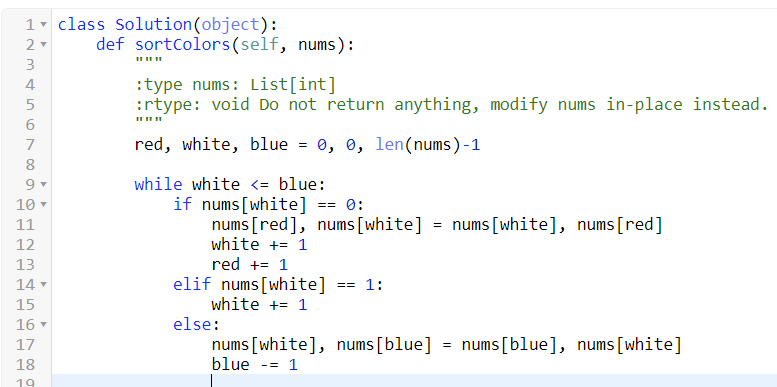
最简单的思想：排序

不过sort和sorted存在区别：



若不能使用排序：

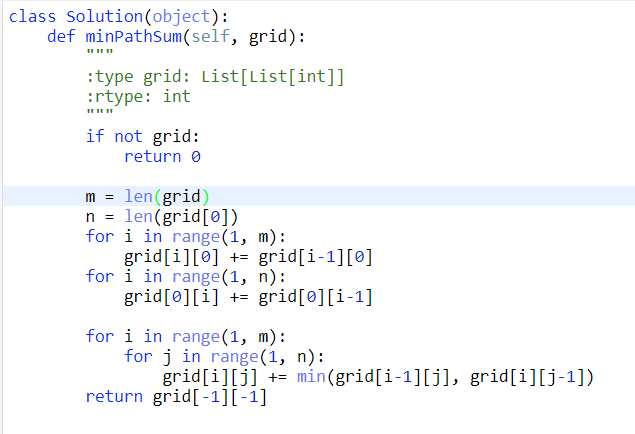
则是一个荷兰的分区问题。我们将数组分为四组:红色、白色、未分类和蓝色。最初，我们将所有的元素归类为未分类。只要白色指针小于蓝色指针，我们就从开始迭代。如果白色指针是红色(nums[white]== 0)，我们就会用红色指针交换，并将白色和红色指针移到前面。如果指针是白色的(nums[white]== 1)，元素已经在正确的位置，所以我们不需要交换，只需将白色指针向前移动。如果白色指针是蓝色的，我们将与最新的未分类元素交换。



**20. 最短路径和（题号64 Minimum Path Sum）**

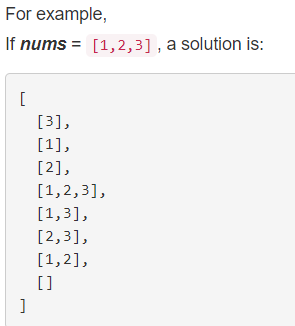
题目：给定一个m\*n的grid，里边充填非负整数，从左上开始，到右下结束，只能向右和向下移动，求出最小的和

思路：有点类似于18题求路径的数量，还是四个格一组，先求向右的和，再求向下的和，最后取向左和向上的最小值即可



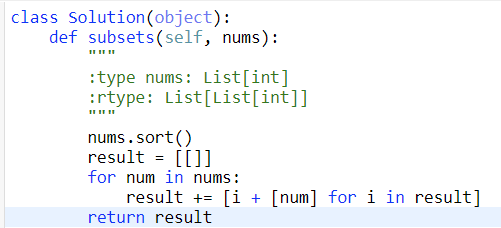
**21. 子集（题号78，Subsets）**

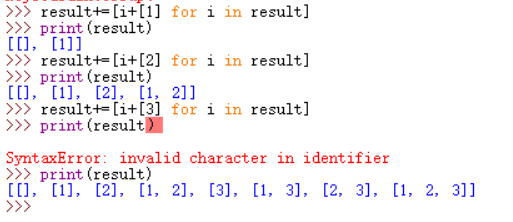
题目：求给定数组所有的子集



思路：保留已有子集的同时，给之前的每个子集多加一个新树，形成新的子集

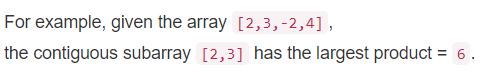
（排序可省略），for在后表示列表生成式，相当于列表.append



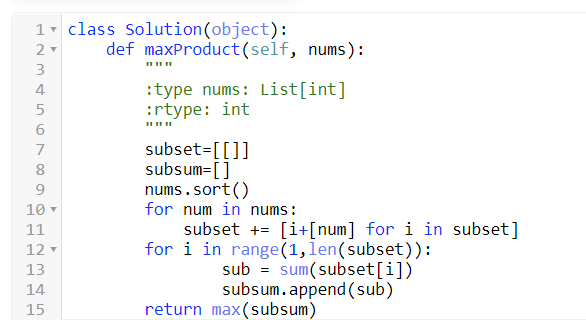


**22. 最大产出子集（题号152，Maximum Product Subarray）**

给定数组，求相邻两个数的最大乘积



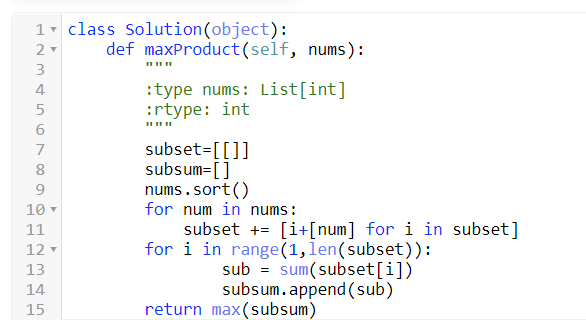
理解错误的方案：以为求和最大的子集



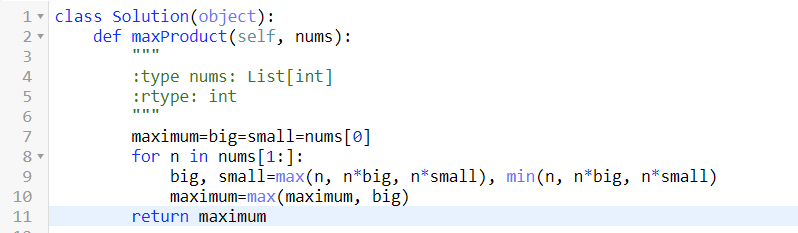
参考上一题的方案，先找出所有子集再求和最大

但此题应求相邻的积最大

我的方案：直接求最大相邻的积



但是，相邻不限个数乘积最大

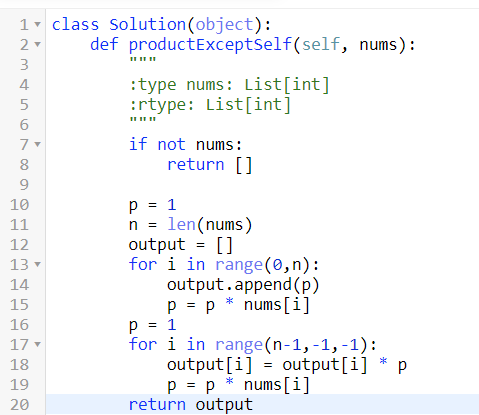
small的出现是为了防止负负得正的情况，为了更新big服务的，很巧妙

**23. 除自身外的乘积数组（题号：238， Product of Array Except Self）**

题目：求数组任意组合的乘积，自身除外



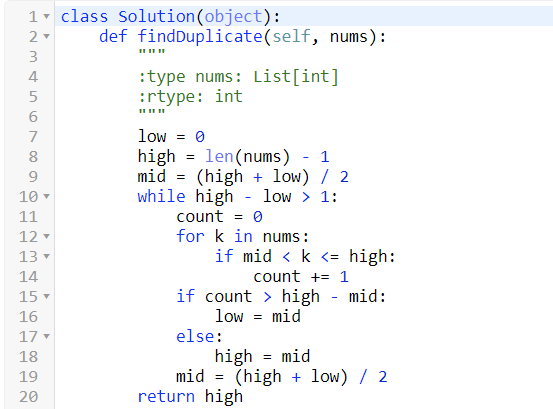
未解决思路：



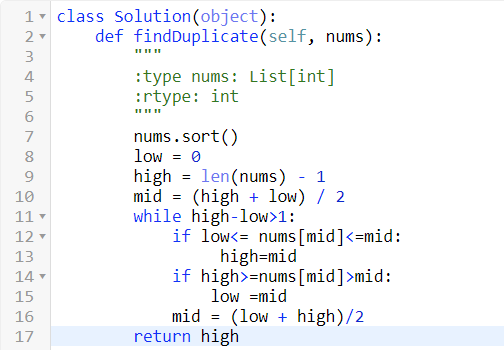
**24. 寻找重复数组（题号287，Find the Duplicate Number）**

题目：给定一个n+1维数组，含有1到n范围内所有整数，找出其中重复的数字，要求时间复杂度在O（n2）以内

思路：二分法查找，先设定一个计数器count，遍历数组中的数，检验后半部分大于mid的数字个数，用count计数，若它大于后半部分high-mid的范围，说明后边的数字有重复的，把low设置到原mid位置重来，若不大于，则说明在前半部分，把high设到原来mid位置重来，最终high和low相邻时，high即为重复的数字



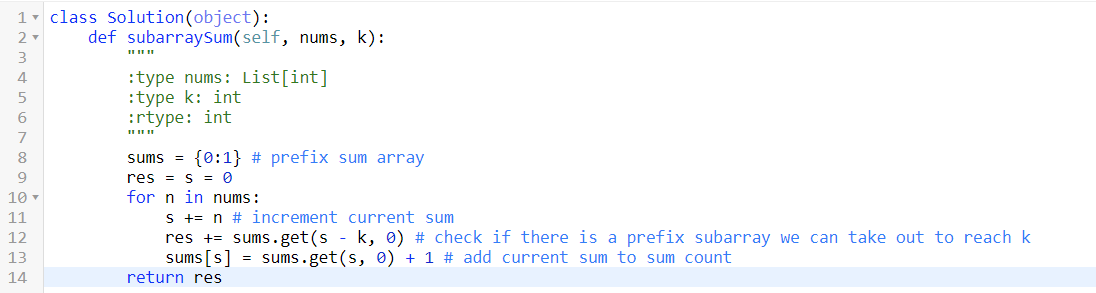
自己写的方案，更简化一点，直接找num[mid]的范围



**25. 寻找和为K的子数组（题号560，Subarray Sum Equals K）**

题目：给定一个数组和一个整数k，求相邻数相加为k的子集个数

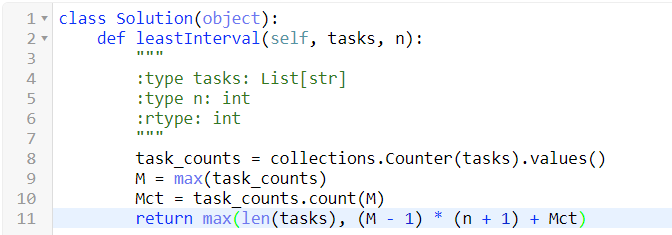
操作：创建一个字典sums，初始化sums={0:1}（便于初次达到k的计数），遍历数组按顺序加和，并每次检查是否当前和是否能与之前的某个和之和达到k，若是则计数res+1，若不是res+0，随后创建当前和的字典，即给当前和的计数+1（便于检查后续是否达到k）

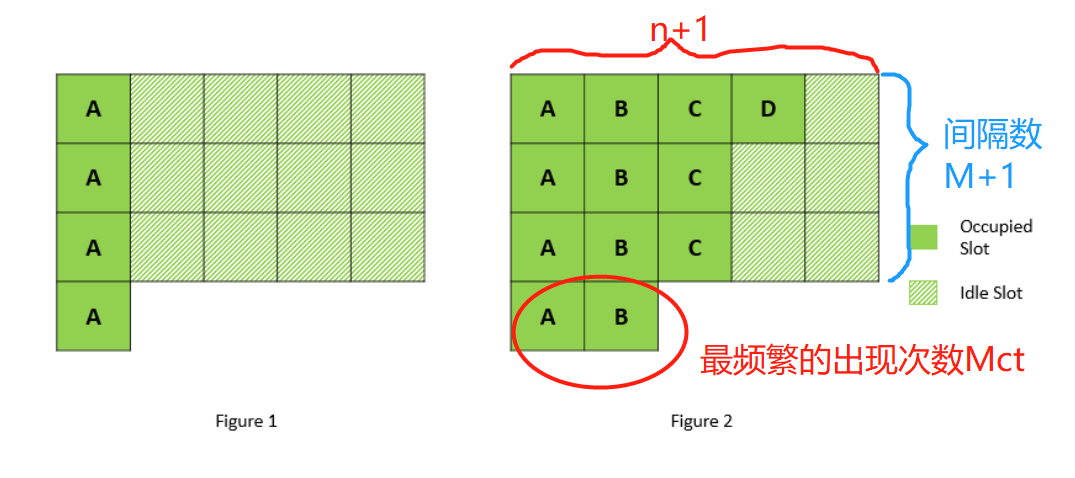


**26. 任务时间（题号621，Task Scheduler）**

题目：CPU执行标号为A-Z的任务，每个任务需要1个时间完成不同字母代表不同任务，CPU执行两个标号相同的任务时需要间隔n个时间，不同标号不需间隔时间，求给定训练序列与规定空闲时间n时，所需最少的CPU时间

思路：本题的关键在于想明白任务时间与标号内容无关，而最小标号时间取决于重复率最高的任务，找出重复率最高的任务，使之隔开固定的时间n，随后往里依次插入频率降低的任务，最后算总长度即可





Counter类用于计数，用法如下：

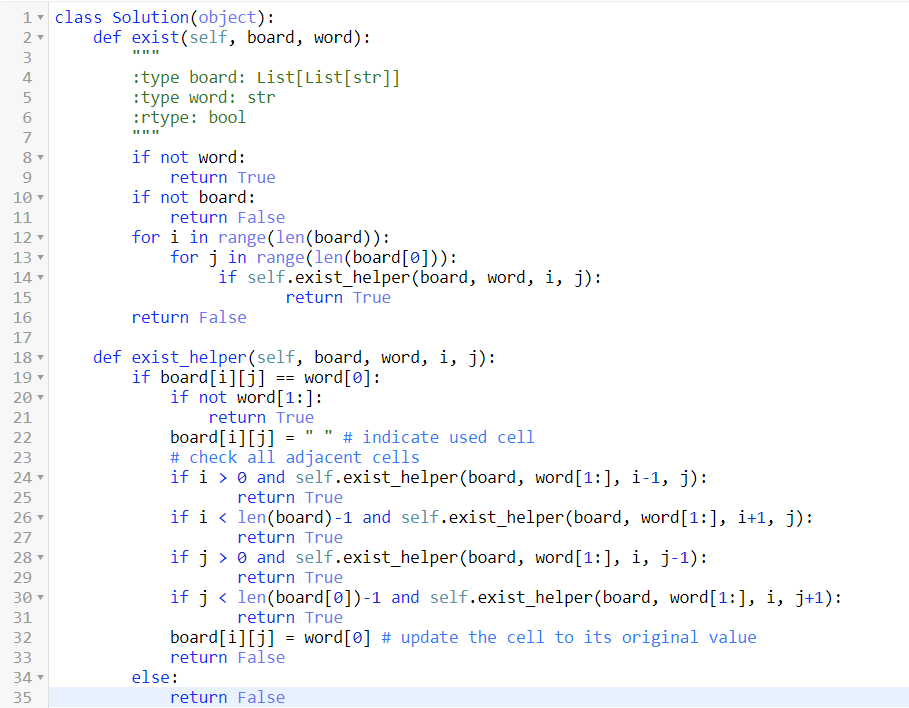


**27.搜索单词（题号79，Word Search）**

题目：给定词板数组，给定目标单词，在词板中上下左右寻找，每个位置只可用一次，若能找出返回为True,不能为False

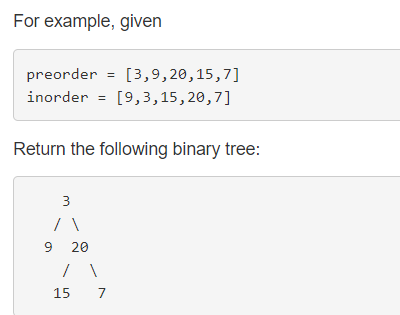
****

思路：锁定一个位置，上下左右遍历，如有与下一个字母吻合的，则循环继续，没有返回False



**28. 二叉树的构造（题号）**

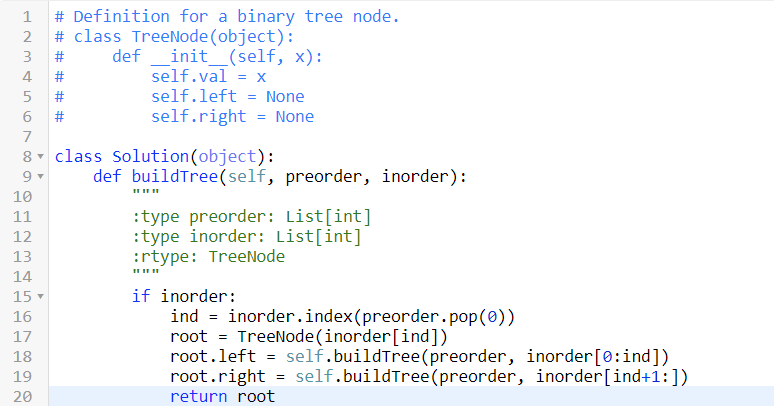
题目：给出前序遍历和中序遍历数组，构造相应的二叉树



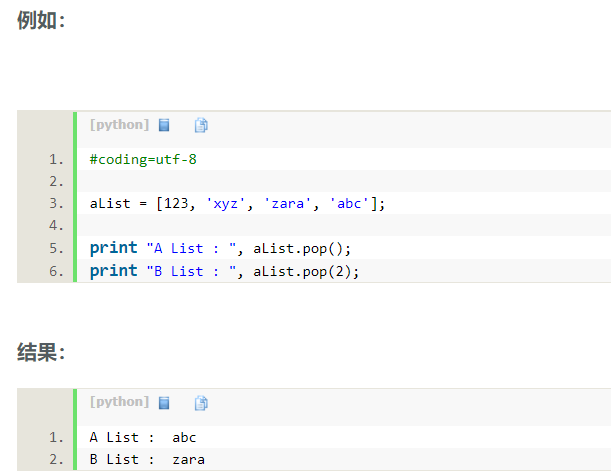
补充：遍历算法

前序Preorder：根左右；中序Inorder：左根右；后序Postorder：左右根

思路：定义二叉树函数TreeNode，左右结点分别引定义函数，函数的调用过程其实是一种嵌套的循环

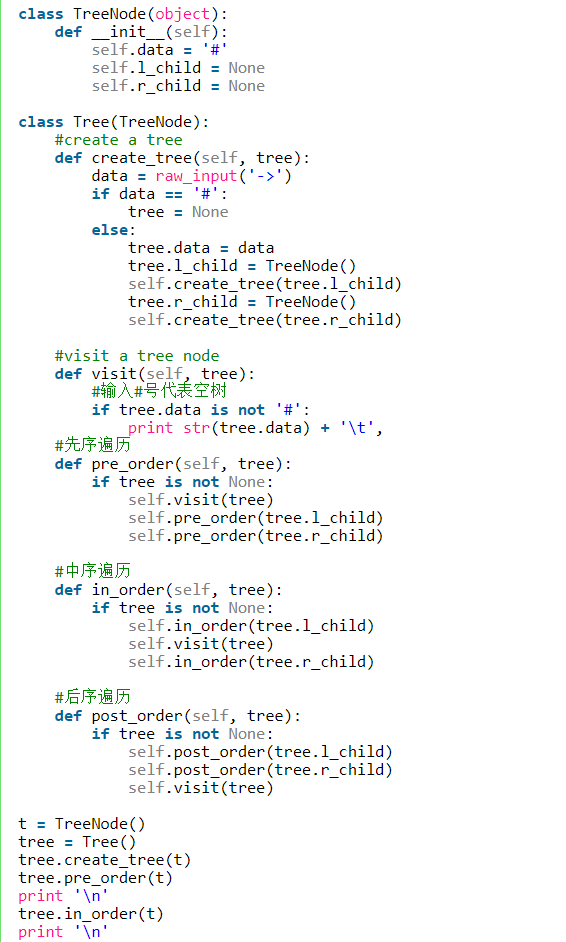


Pop()函数是返回数组删除的项，此时原数组也发生变化



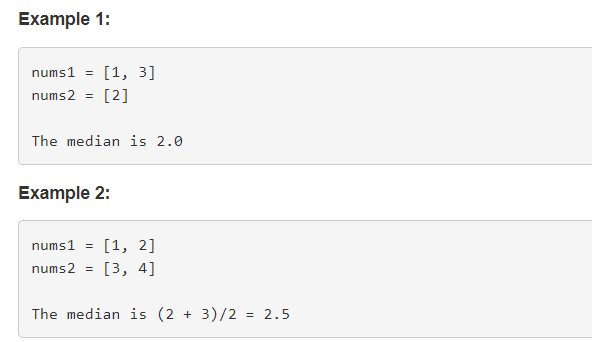
Index()函数是返回数组值的位置

补充：二叉树的创建与遍历代码



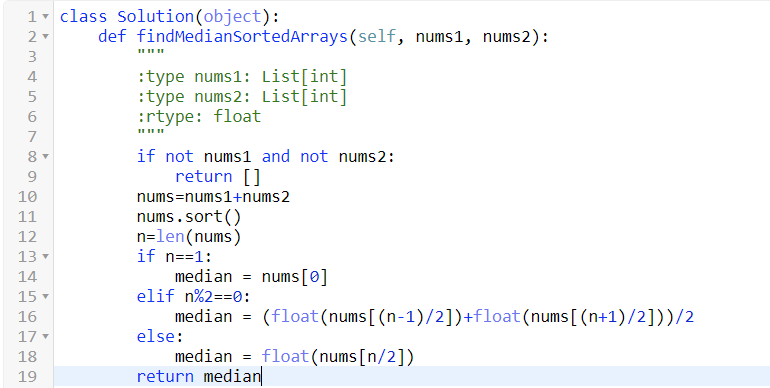
**29.找两个数组的中位数（题号4：Median of Two Sorted Arrays）**

题目：给定两个数组，求两个数组的中位数



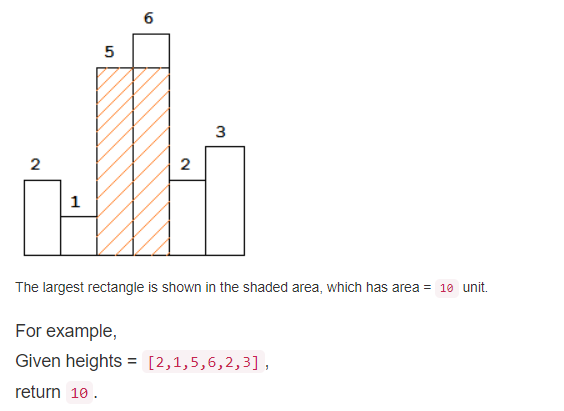
思路：把两个数组合并，然后排序，分奇偶数讨论中位数的计算

要注意：两个数组全为空返回应为空，而两个数组有一个为空，一个长度为1时，返回应为长度为1的数组

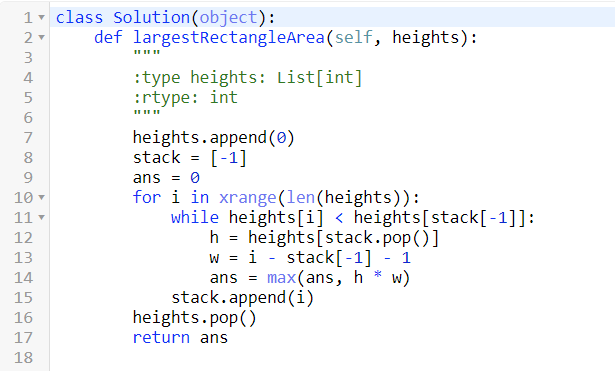


30. 直方图中的最大矩形（题号84，Largest Rectangle in Histogram）

题目：给定数组，求其形成的直方图中最大矩形的面积

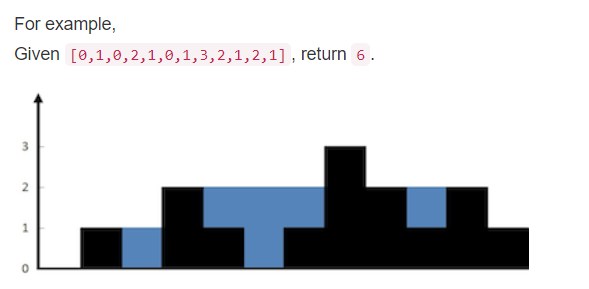


思路：



**31.装雨水（题号42，Trapping Rain Water）**

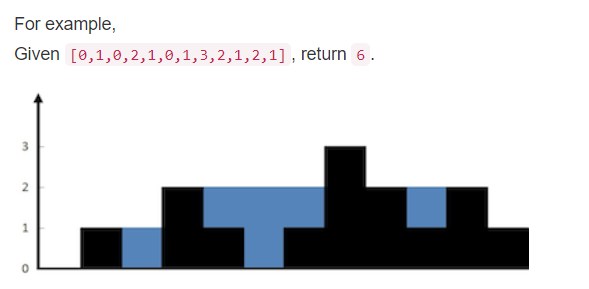
题目：给定一个数组，求数与数之间组成的容器，能装多少雨水



思路一：直接解法

对于每个点来说，它能装的水量等于它左右两边最大值的最小值，减去它自身的高度

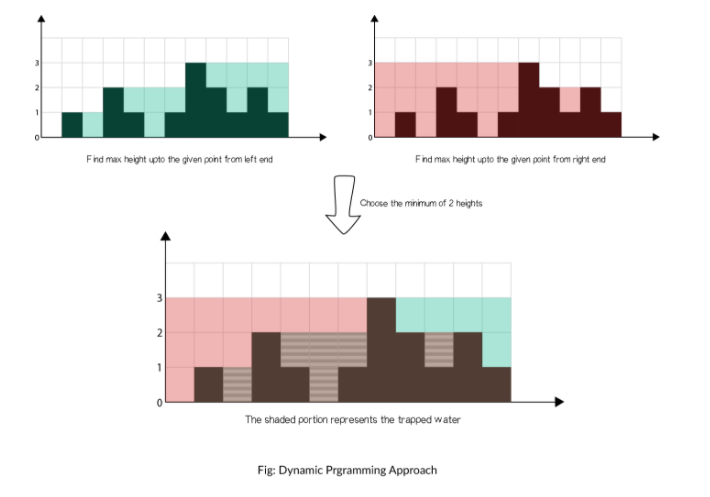
即，大循环循环每个点，小循环对于每一个点分别找它左右两边的最大值，取其中的最小值，然后减去自身的高度

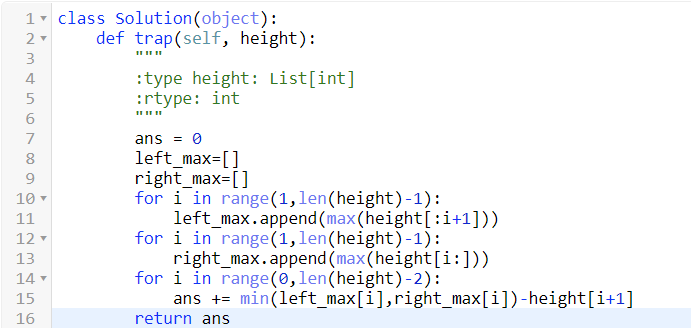


时间复杂度：O(n2)，对于每个点，循环了它的左右两侧，因此超时

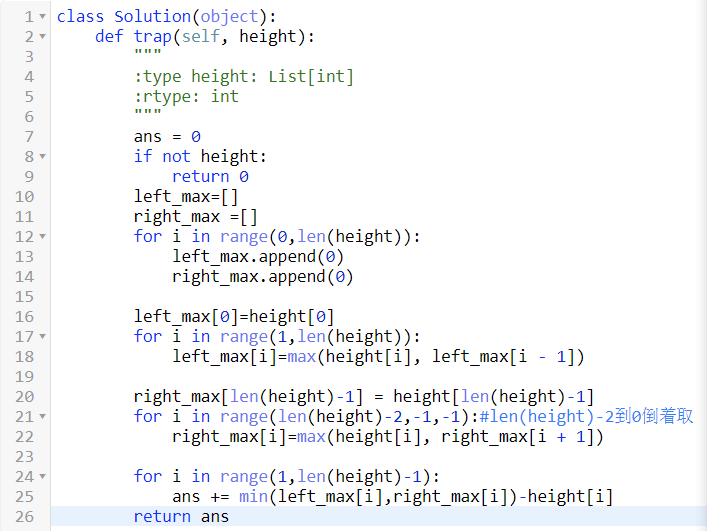
空间复杂度：O(1

思路二：将思路一中两边的最值储存起来，就可以降低算法复杂度





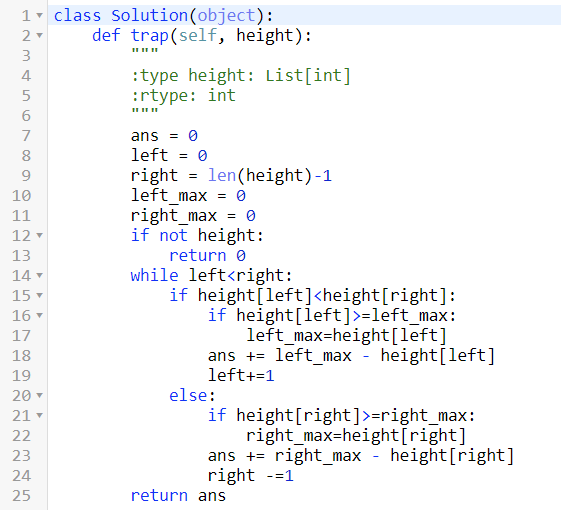
但是这种方法还是超时，因为找两边最值时，max的算法复杂度是n，因此循环后可能又达到了n^2，因此采取以下方法，即两两比较，取大的，也就是每次所得的数一定是前边所有数里最大的了，因此新的数只需要和它前一个比即可，需要注意的主要是，取左边最大时顺着取即可，而取右边最大需要逆着取，重点在第21行



此时算法时间复杂度为n，空间复杂度也是n

思路三：用两个指针

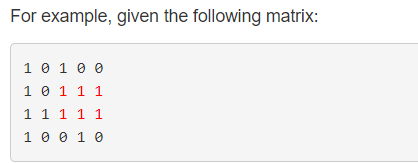
上两个思路都是分别找的左右两边的最值，其实可以同时找，一个指针从左边开始，一个指针从右边开始，如果左边的当前值小于右边的当前值，那么左边这个点的储水量一定取决于左边的最大值，因为之后右边的最大值一定还会大于左边的当前值，同理，如果右边的当前值小于左边的当前值，那么右边这个点的储水量一定取决于右边的最大值，因此找到一侧最大值时即可将它加入总面积中，所以两边同时找，不断更新最大值，也相当于从两边一起遍历这个数组，当左边的指针与右边的相遇时，所有点的储水量就都找完了，这样空间复杂度是n，时间复杂度是1，最简便



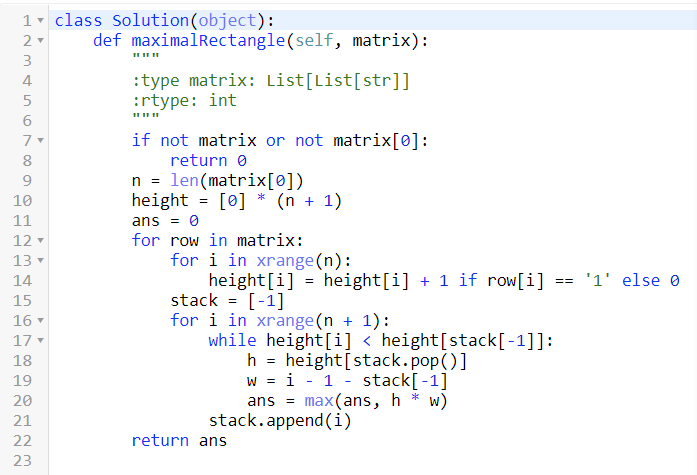
思路四，栈（solution有，待续）

**32. 最大矩形（题号85,Maximal Rectangle）**

题目：给定二维二进制矩阵，求出里边所有元素均为1的最大矩形

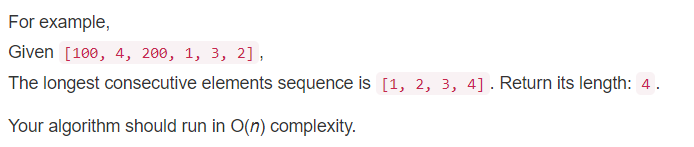


思路：同30最大矩阵



**33.最大连续序列（题号128，Longest Consecutive Sequence）**

题目：给定一个数组，求出它最大连续数字的个数



思路：把数组排序，然后把每个数和它前一个数比较，如果等于前一个数，对连续数列计数不影响，因此不处理，若不等于前一个数，则判断是否等于前一个数+1，若是则计数+1，若不是则返回此前计的最大数，将当前计数置1以便下次计数

