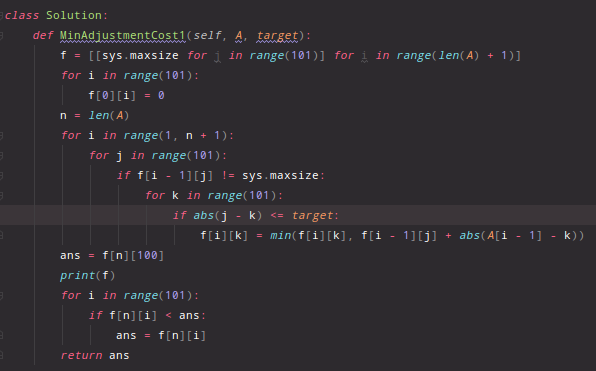
# 动态规划

### 1、最小调整代价

给定一个整数数组，调整每个数的大小，使得相邻两个数的差不大于一个给定的整数target，调整每个数的代价为调整前后差的绝对值，求调整代价之和的最小值。

例如：对于数组【1 4 2 3】和target = 1，最小的方案是调整为【2,3,2,3】，调整代价之和为2，返回2。

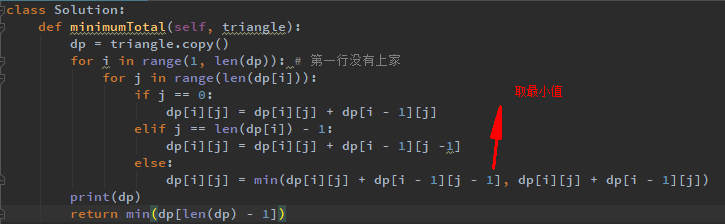
我的思路：既然是求最小值，就大概率要用到动态规划，建立一个动态数组dp[i]表示调整前i个数所需的最小调整代价和。



### 动态规划：数字三角形

给定一个数字三角形，每个位置的数字代表要走的步数长度，找到从顶部到底部的最短路径和，每前进一步则移动到下面相邻一行的数字上。

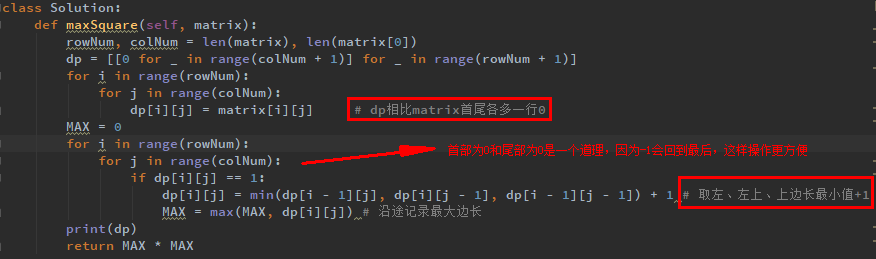
我的思路：从顶点到底部，路径有很多条，但是起点只有一个，终点只有最后一行数字，那么建立一个数组dp[i][j]代表的是从起点到达该点的最短路径。dp[i][j]又分为三种类型：左边缘点，中间点和右边缘点。



### 动态规划：最大正方形

在一个元素为0和1的二维矩阵中找到全为1的最大正方形。

我的思路：设定一个动态数组dp[i][j]，表明以这个点为右下角的最大正方形的边长。将找最大正方形的问题转化到找最大正方形的边长的问题。如果该点为1的话，某点的最大正方形边长等于min（该点上方最大正方形边长，该点左方最大正方形边长，该点左上方最大正方形边长）+ 1，如果该点为0， 那么还会是0，因为无法以该点为右下角找到正方形。

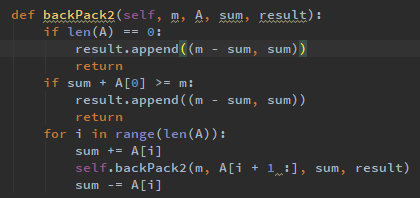


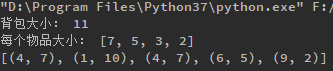
### 动态规划：背包问题

给定背包的大小为m，物品的数量为n，每个物品的大小为A[i]，求出背包最多能装满的空间大小。

给定4个物品【2,3,5,7】，如果背包的大小为11，则可以选择2,3,5装入背包，最多可以装满10的空间；如果背包的大小为12，则可以选择2,3,7装入背包，最多可以装满12的空间。

我的思路：明显该用贪心思想，先将大的往背包里放，将数组倒序排列，从大到小回溯物品的重量，跳出条件是所有物品放完，和加入的物品的重量超过背包的容量，但这里依旧没有问所有可能放的情况，所以的话还是用动态规划，还有就是用回溯不好把重量放入数组



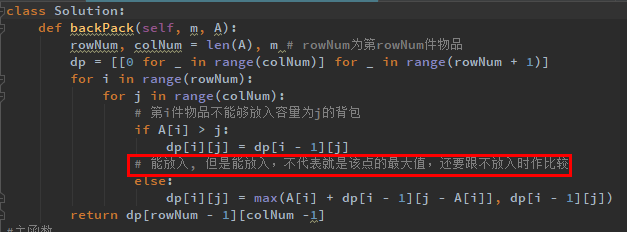


分别对应着背包中装[7],[7,3],[5,3],[3,2],[2]的情况。太复杂 。

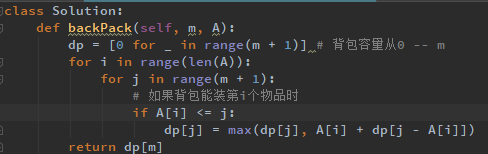
解题思路：该题是0/1背包问题，用动态规划，设dp[i][j]表示背包容量为j时，装前i个物品能装的最大重量，且每件物品只能装一次。

公式：1.第i个物品不能装时： dp[i][j] = dp[i - 1][j]

2.第i个物品能装时：dp[i][j] = max(A[i] + dp[i - 1][j - A[i], dp[i - 1][j])



上述是二维解法，下述是一维解法。



#### 拓展：三种基本背包问题：

##### 1、0/1背包问题:

问题描述：有n件物品和容量为m的背包 给出i件物品的重量以及价值 求解让装入背包的物品重量不超过背包容量 且价值最大 。其中物品重量数组为w，价值数组为v，每件物品只能选一次。  
 特点:这是最简单的背包问题，特点是每个物品只有一件供你选择放还是不放。

解题思路:  
 ① 二维解法  
 设f[i][j]表示前 i 件物品 总重量不超过 j 的最大价值，那么当i = n ，j = m, 就表示前n件物品，总重量不超过m的最大价值。可分为两种情况，能放第i件物品的价值和不能放第i件物品的价值，得出更大值即可，可得出状态转移方程：

1. 当第i件物品能放时：f[i][j] = max{v[i] + f[i - 1][j - w[i]], f[i -1][j]}
2. 当第i件物品不能放时：f[i][j] = f[i - 1][j]

②一维解法

因为二维解法当数据量很大时会没办法ac，所以直接用一维数组也可。  
 设f[j]表示重量不超过j公斤的最大价值，当j = m时，并循环n轮，就是前n件物品，总重量不超过m的最大价值，也就是说如果用二维数组，只用到上一行的值，并且取值只取最后一个数的时候，那么其实可以反复利用该数组，可得出状态转移方程：

1. 当第i件物品不能放时，f[j]保持不动，表示取前i - 1件物品的最大价值。
2. 当第i件物品能放时，f[j] = max{f[j], f[j - w[i] + v[i]}

##### 2、完全背包问题

问题描述：有n件物品和容量为m的背包 给出i件物品的重量以及价值 求解让装入背包的物品重量不超过背包容量 且价值最大 。其中物品重量数组为w，价值数组为v，每件物品无限选用。  
 特点：题干看似与01一样 但它的特点是每个物品可以无限选用。

① 二维解法

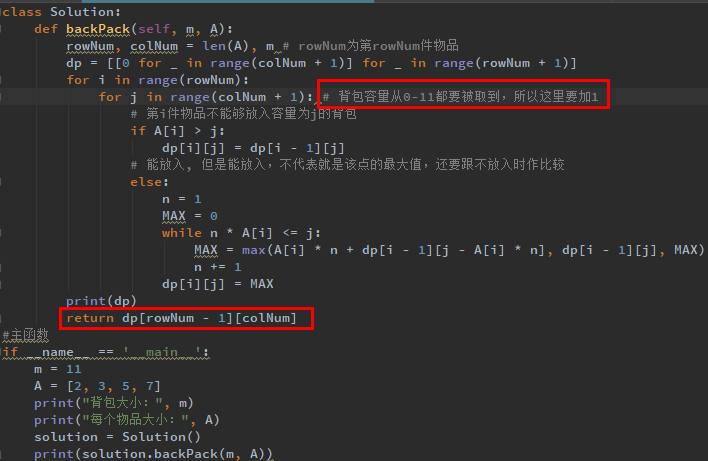
设f[i][j]表示背包容量为j时，能装前i个物品的的最大价值 可得出状态转移方程

1. 当第i件物品不能放时：f[i][j] = f[i - 1][j]
2. 当第i件物品能放时：while n \* w[i] <= j {

MAX = max( n \* v[i] + f[i - 1][j - n \* w[i]], f[i - 1][j], MAX )

}

假如该题该为完全背包，即能无限取用：



②一维解法

f[j] 依旧表示容量为j的背包，能存放的物品的最大价值。

1. 当不能存放第i件物品时，那么存放的最大价值就是存放第i - 1件物品时存放的，那么就保持不变。
2. 当能存放第i件物品时，MAX = 0

while n \* w[i] <= j {

MAX = max{f[j], f[j - n \* w[i]] + n \* v[i], MAX}

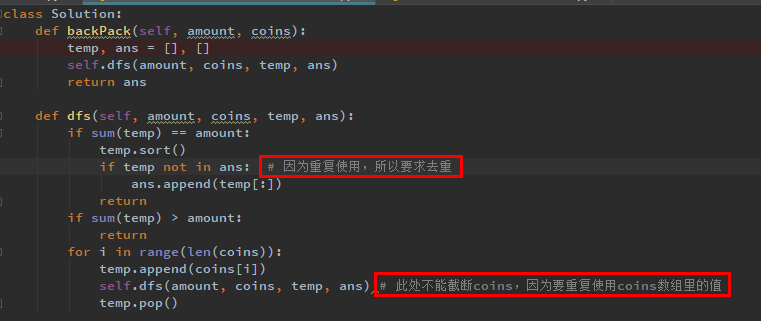
}

f[j] = MAX

再例：给定不同面额的硬币和一个总金额。写出函数来计算可以凑成总金额的硬币组合数。假设每一种面额的硬币有无限个。

如：amount = 5, coins = [1, 2, 5]

解题思路：定义一个数组f[i][j], 表示当总金额为j时，用前i个硬币，能凑成j的情况数。但是这道题不是求最值，其实用回溯更好一点，回溯其硬币的种类，先放大的，再放小的。但是存在重复使用，需要去重，代价很大。



##### 3、多重背包问题

问题描述：有n件物品和容量为m的背包 给出i件物品的重量以及价值 还有数量 求解让装入背包的物品重量不超过背包容量 且价值最大 。  
 特点 ：它与完全背包有类似点 特点是每个物品都有了一定的数量。数量数组为amount[i]

① 二维解法

状态转移方程为：

1. 当第i件物品不能放时：f[i][j] = f[i - 1][j]
2. 当第i件物品能放时：while n <= amount[i] and n \* w[i] <= j { # 注意数量多意味着可能 会超出背包容量。

MAX = max( n \* v[i] + f[i - 1][j - n \* w[i]], f[i - 1][j], MAX )

}

②一维解法

f[j] 依旧表示容量为j的背包，能存放的物品的最大价值。

1. 当不能存放第i件物品时，那么存放的最大价值就是存放第i - 1件物品时存放的，那么就保持不变。
2. 当能存放第i件物品时，MAX = 0

while n <= amount[i] and n \* w[i] <= j {

MAX = max{f[j], f[j - n \* w[i]] + n \* v[i], MAX}

}

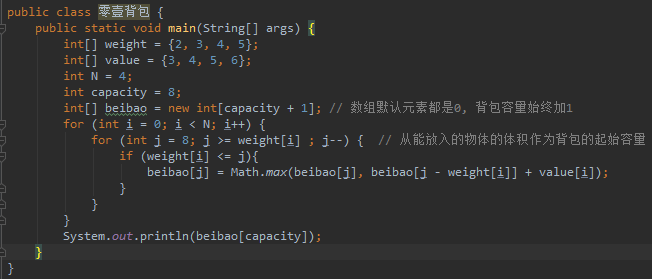
f[j] = MAX

#### 动态规划入门到熟悉

<https://blog.csdn.net/hebtu666/article/details/100585136>

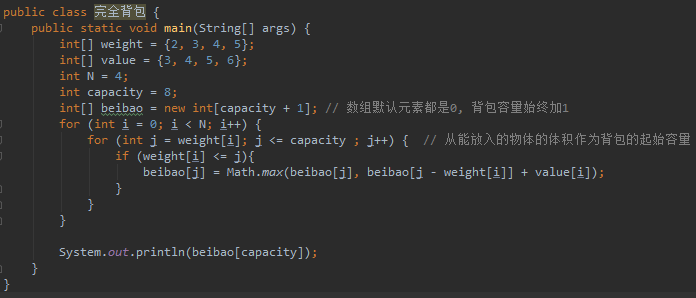
##### 0/1背包问题：

思路：因为每个物体只能取一次，且用的是一维数组，所以背包容量从大到小，因为这样能保证数组左边的元素是前面的的物体组成的和，与该物体无关，这样当放入该物体后，求剩余背包容量所能放置的最大价值，且背包容量到能够放入该物体即可，无需到0.



##### 完全背包问题：（每个物体可以去无限个）

思路：相比于0/1背包，其实只需将第二个循环的顺序调转过来即可。好好体会。



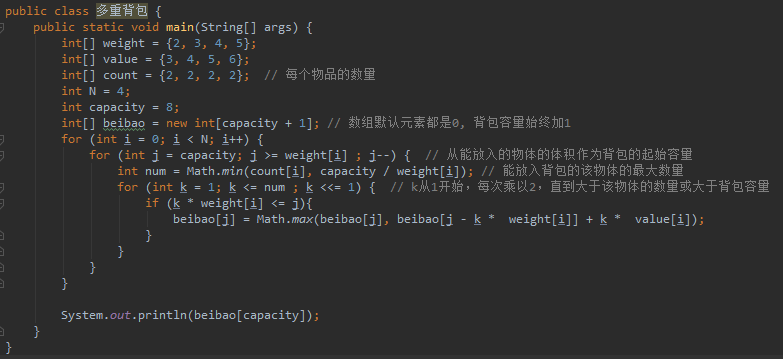
##### 多重背包问题：（每个物体只能取有限个）

思路：1、将多个物体拆成一个一个物体，用01背包求解

1. 将多个物体拆成二进制数的物体堆，用01背包求解

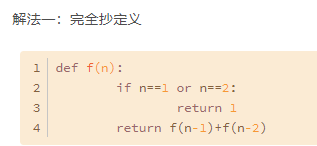
举例：15个物体拆成15个，可以将1-15个物体放入背包，15个物体才成1、2、4、8四堆，同样可以表示成1-15个物体放入背包。

注意：虽然需要上面的数进行组合，但是还是要从大到小，因为1、2、4、8不能重复取数。

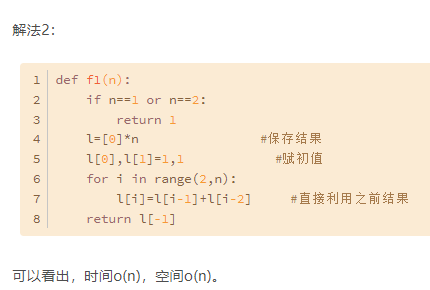


##### 斐波那契数列系列问题：

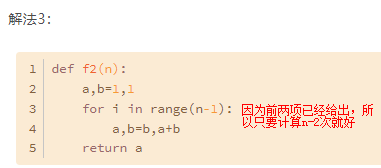
**思路：如果只是求第n项，以下三种方法，从坏到好：**



该解法效率极低，因为每次求一个f（n）都要求出f（n...1）, 出现了大量的重复计算



该解法保存了中间值，避免了重复计算，但如果只需要求出第n项，那么是不是空间有点浪费呢？所以有了下面的解法：



**青蛙跳台阶和爬楼梯问题：（只能跳、爬一阶或两阶）**

思路：代码只是将刚开始的两个值从1,1改为1,2。

动态规划思想：到第n阶的情况只有两种情况：从第n-1阶迈一步；从第n-2阶迈两步

所以f（n） = f（n - 1） + f（n - 2）

**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？**

动态规划思想：覆盖到长度为n的地方有两种情况：

在第n个位置竖着放一个小矩形；

从第n-1个位置横着放两个小矩形；

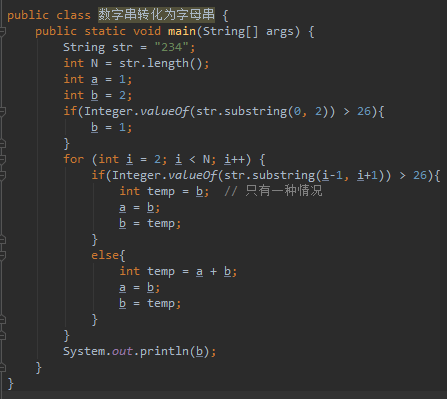
所以f（n） = f（n - 1） + f（n - 2）

**给定一个由0-9组成的字符串，1可以转化成A，2可以转化成B。依此类推。。25可以转化成Y，26可以转化成z，给一个字符串，返回能转化的字母串的有几种？**

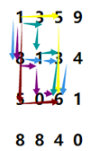
动态规划思想：第n个数字字符转换完成字母分为两种情况：

第n个位置，也就是自身转化为字母

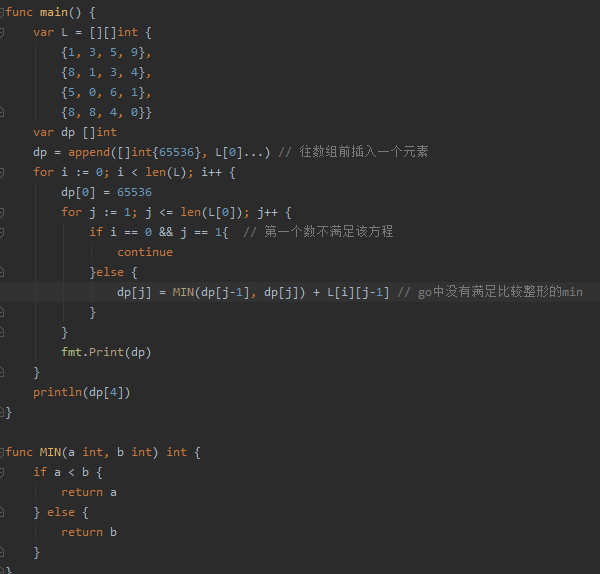
结合第n-1个位置，转换为字母（两位数不能大于26）



## 矩阵走路问题

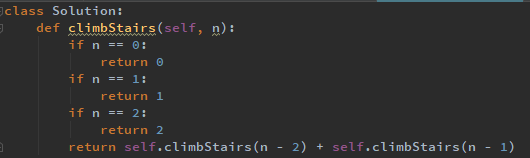
从矩阵L左上角到右下角经过的路径的最小和

状态转移方程：dp[j] = min(dp[j-1], dp[j]) + L[i][j]。



### 动态规划：爬楼梯

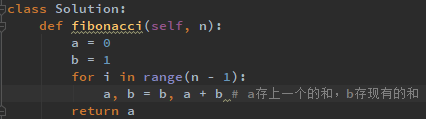
假设正在爬楼梯，需要n步才能到达顶部，但每次只能爬一步或者两步，求出爬到楼顶的方法总数。



### 动态规划：斐波那契数列

查找斐波那契数列中第N个数，所谓斐波那契数列是指：前两个数是0和1；第i个数是第i - 1和第i -2个数的和；前十个数是：0、1、1、2、3、5、8、13、21、34.

爬楼梯的前两个数是1和2.

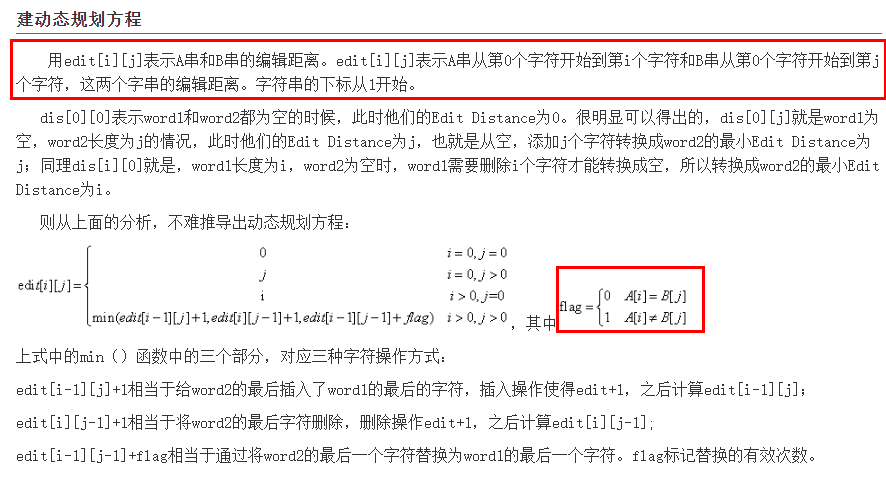


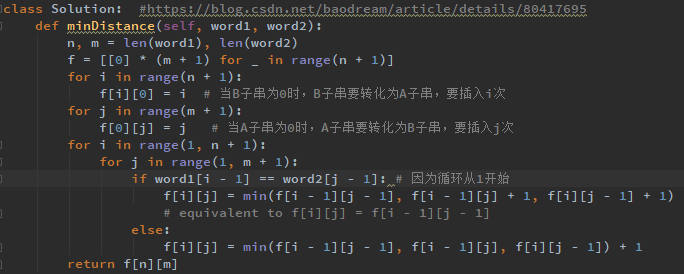
### 动态规划：编辑距离

给定两个单词word1和word2，计算将word1转换为word2的最少操作次数。总共有三种操作方法：

1. 插入一个字符
2. 删除一个字符
3. 替换一个字符

我的思路：无





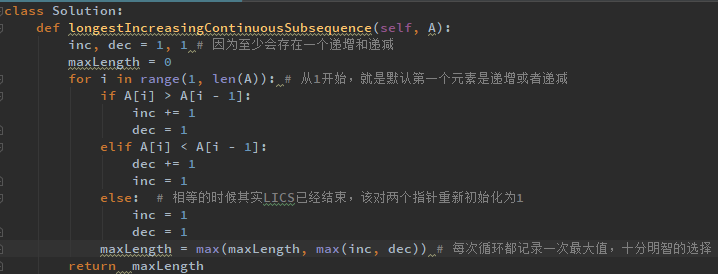
### 动态规划：最长上升连续子序列

给定一个整数（数组长度为n， 下标为[0, n - 1]），本例将找出该数组中最长的上升连续子序列（LICS）， LICS定义为按照元素大小从右向左或从左到右排列的序列。

我的思路：由最长上升子序列可知，dp[i]表示前i个元素中的最长上升子序列。是否可以用类似的方法进行呢？答案是不能

解题思路：因为递增和递减都称作为LICS，所以需要两个指针，且因为LICS不会存在交叉，那么其实可以利用中间初始化的方式进行求解，复杂度就是O（n）

关键思路在于：当不增不减后，增指针和减指针要清1.

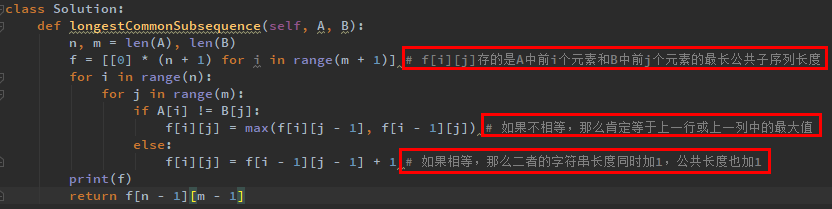


### 动态规划：最长公共子序列

给定两个字符串，找到最长公共子序列（LCS），返回LCS的长度。最长公共子序列定义为：一组序列（通常2个）的最长公共子序列。注意，不同于子字符串，LCS不需要是连续的子字符串。

我的思路：肯定要用动态规划

解题思路：定义一个二维数组dp[i][j]，第一个数组的前i个位置和第二个数组的前j个位置的最长公共子序列。



### 动态规划：最长上升子序列

给定一个整数序列，找到最长上升子序列（LIS），返回LIS的长度。其中，LIS是指一个序列中最长的单调递增的子序列。

我的思路：因为不是连续的，所以其实是可以用回溯的，找到所有的可能的情况。出口就是到了末尾，当找到递增的元素就进入下一层递归，找不到就continue，但这里只是要求长度，所以用的肯定是动态规划，如果要求所有可能情况，用回溯的概率就大一点。

解题思路：定义一个数组dp，记录i和i位置之前的最长上升子序列。

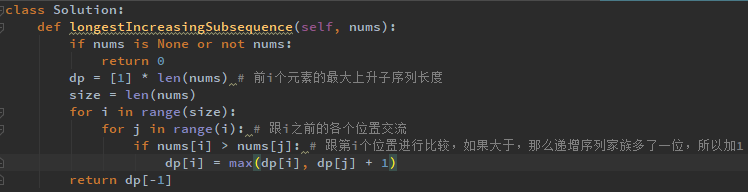
设dp[i]: 前i项的最长上升子序列的长度

for j in range[1,i]:

if dp[i] > dp[j]:

dp[i] = max(dp[i], dp[j] + 1)

Return dp[n]



### 不同的子序列

给定字符串S和字符串T，计算在S中不同子序列T出现的个数。子序列字符串是原始字符串通过删除一些（或0个）字符产生的一个新字符串，并且对剩下字符的相对位置没有影响。例如，“ACE”是“ABCDE”的子序列字符串，而“AEC”不是。

我的思路：这题如果问的是情况的话，毫无疑问我会选择用回溯，回溯的范围是两个字符串的长度之差，删除指定长度的字符。但明显还是要用动态规划，因为它问你的是个数。

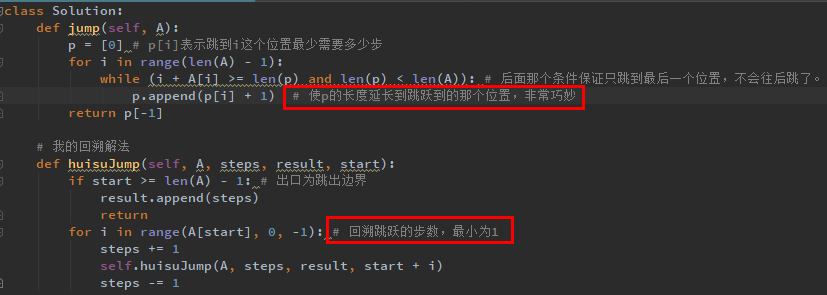
解题思路：还是要建立一个dp[i][j]，表示S中的前i个元素中T中的前j个元素的个数。

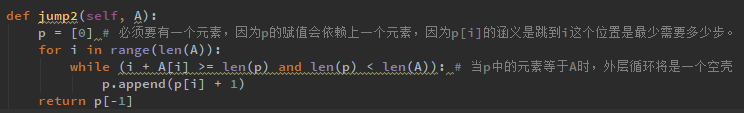
### 动态规划：跳跃游戏2

给定一个非负整数数组，最初定位在数组的第一个位置。数组中的每个元素代表在那个位置可以跳跃的最大长度，本例将使用最少的跳跃次数到达数组的最后一个位置，并返回跳跃次数。

我的思路：用回溯解决， 回溯跳跃的步数。

解题思路：定义一个p[i]，p[i]表示跳到i这个位置最少需要多少步





### 动态规划：打劫房屋1

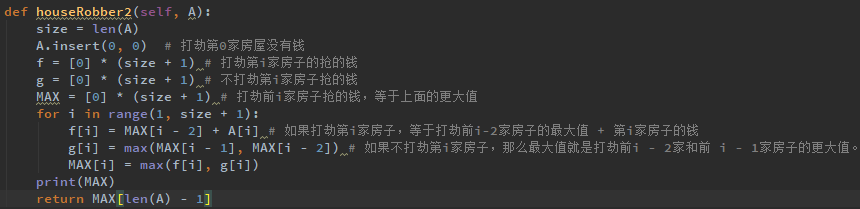
假设一个专业的窃贼准备沿着一条街打劫房屋，每个房子都存放着特定金额的钱。面临的唯一约束条件是：相邻的房子装着相互联系的防盗系统，且当相邻的两个房子同一天被打劫时，该系统会自动报警。

给定一个非负整数列表，表示每个房子存放的钱，如果窃贼今晚去打劫，则在不触动报警装置的情况下，求最多可以得到多少钱。

我的思路：要是我的话肯定又用回溯，回溯的是跳跃的步数，步数大于2小于length - 1就好，出口是跳到了最后一间房屋。但这里问的还是一个数，所以肯定用动态规划，联想一下买股票的问题，沿途记下最大值，但是呢这又不是一次买卖的问题，也不是多次买卖取差值。

解题思路：用一个数组f存打劫第i家房子抢的钱，用另一个数组g存不打劫第i家房子抢的钱，用另一个数组MAX存打劫前i家房子所得的最大值，也就是前两者的最大值。

公式：MAX [i] = max(f(i), g(i)) f(i) = MAX(i - 2) + A[i] g[i] = max(MAX(i - 1), MAX(i - 2))

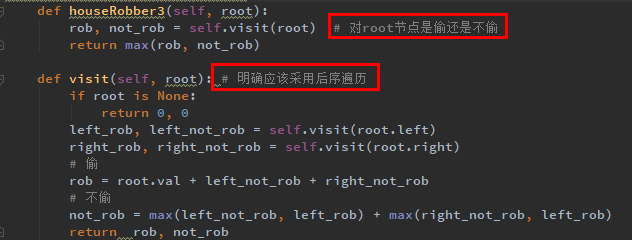


### 动态规划：打劫房屋2

假设打劫的地方是一棵二叉树，每个房子都存放着特定金额的钱。面临的唯一约束条件是：相邻的房子都装着相互联系的防盗系统，且当相邻的两个房子同一天被打劫时，该系统会自动报警。如果窃贼今晚去打劫，在不触动警报装置的情况下，最多可以得到多少钱。

我的思路：如何确定不相邻的通用条件呢？

解题思路：首先，不能偷相邻的两个房子转化成二叉树里的限制条件就是：是否偷这棵树的头节点，如果偷这棵树的头节点，那么他的两颗子树都不能偷，如果不偷这棵树的子节点，那么两棵子树都能偷，每棵子树都要返回其为头结点的偷和不偷的最大值。那么先要进行遍历，那么采用后序遍历的方法，返回的是偷和不偷的最大值。最终比较偷和不偷的最大值。

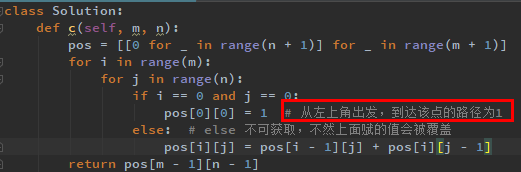


### 动态规划：不同的路径1

有一个机器人位于m行n列的网格的左上角，每一时刻只能向下或者向右移动一步，机器人试图达到网格的右下角，本例将求出不同路径的条数。

我的思路：回溯方向，记录所有情况，得到所有可能路径条数，但是此题不需要算出所有路径，所以用动态规划更合适

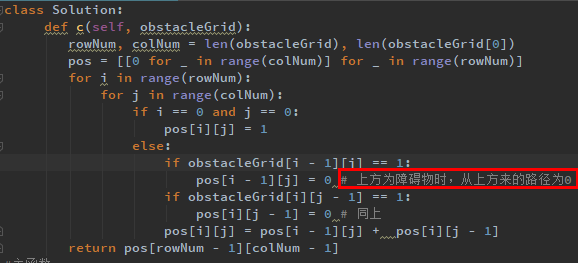
解题思路：因为只有往右往下两个方向，那么到二维网格上的每个点的路径条数都等于上一个路径点的路径条数加上左边一个路径点的路径条数之和，增加一行一列用作第一行第一列之用。**能用该种方法的原因是因为只有两个方向，那么所有的路径都不会重叠。**



### 动态规划：不同的路径2

本例为不同路径的跟进问题，现在考虑网络中有障碍物，求出一共有多少条不同的路径，网络中的障碍和空位置分别用1和0来表示。

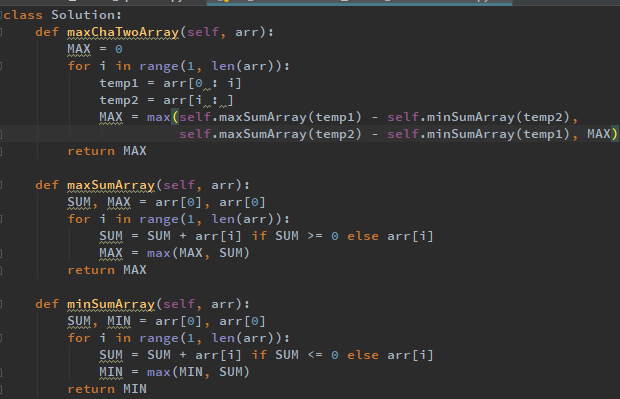
我的思路：相比于上一题，其实就是多了障碍物，其实还是只有两个方向，即向右和向下，那么其实还是一道动态规划问题，如果节点的上面或者左边有障碍物的话路径数清0.



### 动态规划：两个不重叠子数组最大差

给定一个整数数组， 本例将找出两个不重叠的子数组A和B，使两个子数组之间和的查最大，返回这个差值。

子数组最大和子数组最小和的应用。

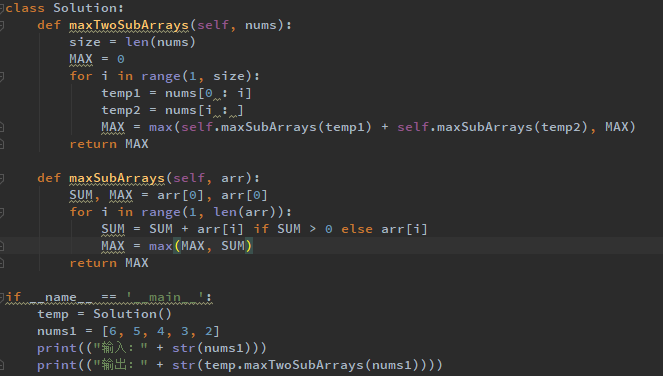


### 动态规划：两个不重叠子数组最大和

给定一个整数数组，本例将找出两个不重叠子数组，是他们的和最大，并返回最大的和。注意，每个子数组应该是连续的。

把数组分为两部分，分别对两部分进行最大和子数组计算，对其中间的位置进行遍历。

注意， 当数组全为正数时是没有意义的。



### 动态规划：最大和子数组

给定一个整数数组，本例将找到一个一个具有最大和的子数组， 返回最大和。

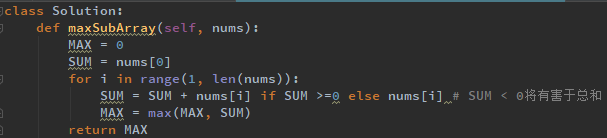
求最大和子数组其实就是不断的寻找子数组起始点的过程，在这个过程中记录其最大和。

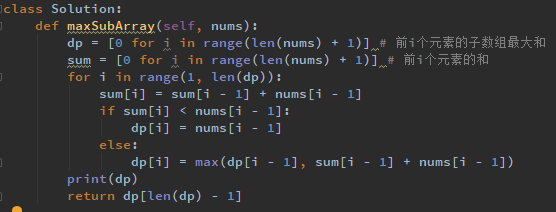
不会存在1-n的和小于2-n的和的情况，因为1必须为负，那么起点就是2了。好好理解。

记住一句话，前n-1项和有害于总和，替换为n。

设dp[i] 前i个元素的最大和

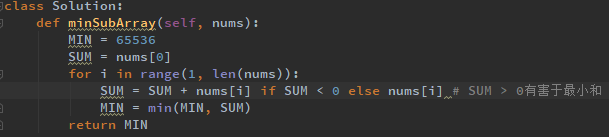
1. 当加上前i - 1个元素的最大和对第i个元素有害时(A[i] > A[i] + dp[i -1]): dp[i] = A[i]
2. 无害时： dp = max(dp[i - 1][j] , dp[i - 1][j] + A[i])





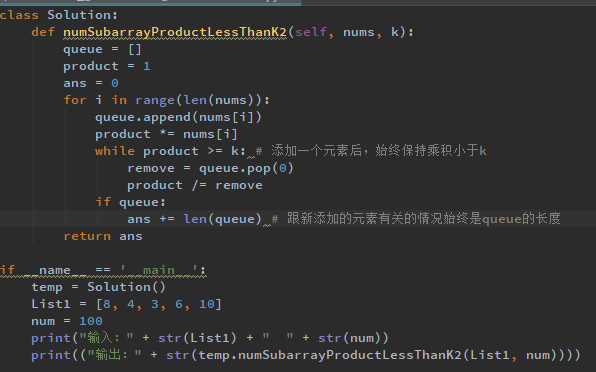
### 动态规划：最小和子数组

给定一个整数数组，本例将找到一个一个具有最小和的子数组， 返回最小和。



### 动态规划：乘积小于k的子数组

利用一个queue，每向queue中添加一个数字，始终保持queue里面存的是乘积小于k的连续数字，这里有一个定理，若往queue里面存入一个数x还小于k，那么将增加len（queue）种与x有关的情况。

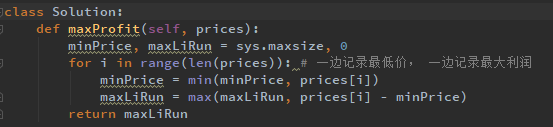


### 动态规划：买卖股票的最佳时机I

假设有一个数组，它的第i个元素是一只给定的股票在第i天的价格， 如果最多只允许完成一次交易（即一次买卖股票），本例将设计一个算法找出最大利润。

我的思路：也就是只能买一次，也只能卖一次，以尽可能低的价买入，以尽可能高的价卖出，将数组分成两部分，遍历中间点，从左边找出最小值，从右边找出最大值。但是这样的事件复杂度是O（n2）

动态规划思路：从左到右扫一遍，一遍记录最小值，一遍记录最大利润

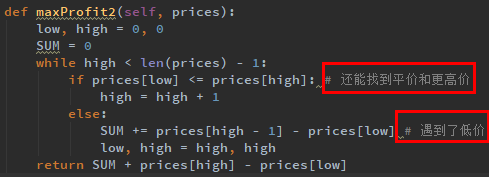


### 动态规划：买爱股票的最佳时机2

假设有一个数组，它的第i个元素是一个给定的股票在第i天的价格，本例将设计一个算法来找到最大利润，可以完成尽可能多的交易（即多次买卖股票），但是不能同时参与多次交易（即在此购买前必须出手股票）

我的思路：根据上题的思路，当达到最大利润时，卖掉并重新计算最低价

解题思路：其实就是贪心的思想，上面的思路其实不需要计算出最大利润再卖掉，而是遇见低价就买入，遇见连续高价的最高价就卖出。



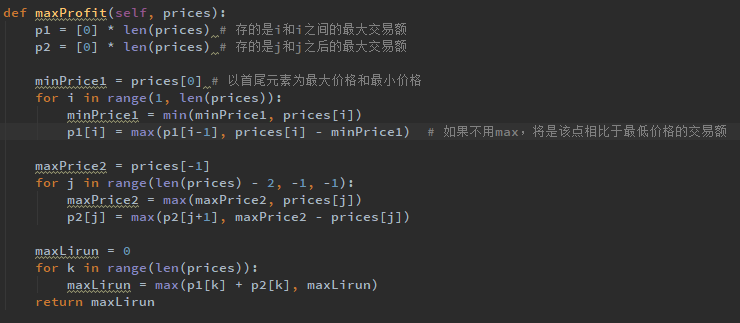
### 动态规划：买卖股票的最佳时机3

假设有一个数组，它的第i个元素时一只给定的股票在第i天的价格，本例将设计一个算法找到最大利润，最多可以完成两笔交易。

我的思路：根据169题的思路，遍历中间点，左右开弓，但是时间复杂度是O（n2）

解题思路：如上思路的话，要创建很多临时数组，这里用的方法是，用p1[i]存储i之前的最大利润，p2[i]存储i之后的最大利润，只需要两个数组对应值相加，进行比较就能得到最大利润，只需向前和向后加起来遍历两遍，总体遍历一遍，也就是三遍。用数组来传递最大和最小值，用来记录该位置之前的最值，是减少复杂度的利器。

注意，因为两次买卖之间存在交集点，只要两者的利润之和足够大，是存在卖出后立即买入的情况的。



### 动态规划：骰子求和

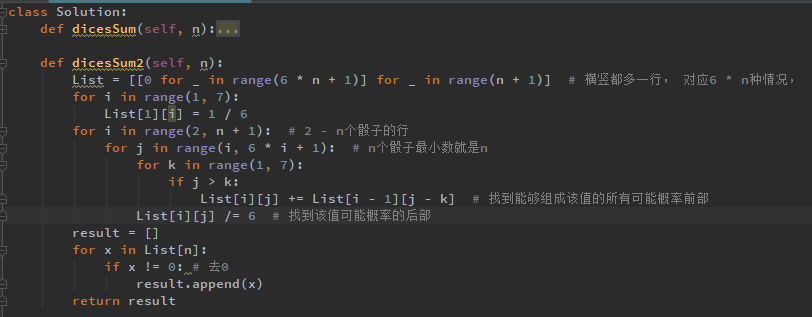
扔n个骰子，向上面的数字之和为S，给定n，本例将列出所有可能的S值即其相应的概率

解题思路：采用动态规划求解，出口肯定是，当只剩下一个骰子时，不论扔出几，概率都是1/6，可以把n个骰子扔出的概率写成一行，每一行存着所有可能出现值的情况的概率，下一行的值基于上一行的值，最后输出第n行的值。

一句话就是把n - 1行当成一个（6 \*（n - 1））个面的骰子，第n个骰子是6个面的骰子，设dp[i][j]，表示前i个骰子，骰出和为j的概率。

状态转移方程为：

1. 当i==1时，概率都为1/6
2. 否则，dp[i][j] = (dp[i-1][j - 6] + dp[i-1][j-5]+....+dp[i-1][j-1])/ 6



### 动态规划：子矩阵和为0

给定一个m行、n列的整数矩阵，元素坐标从左上角的（0，0）开始，到右下角的（m-1,n-1）结束。找到数字总和为0的子矩阵，并返回子矩阵左上角和右下角的编号。

我的思路：就好像要找子数组和为0的情况一样，记录前n项和，当前x项和和前y项和相等时，说明[x + 1, y]这个子数组之和为0。但这里明显不是，因为这里是二维关系。

解题思路：首先这里肯定得用动态规划，如下3维矩阵，这里涉及到两个动态规划：

[[1, 2,3]

[4,5, 6]

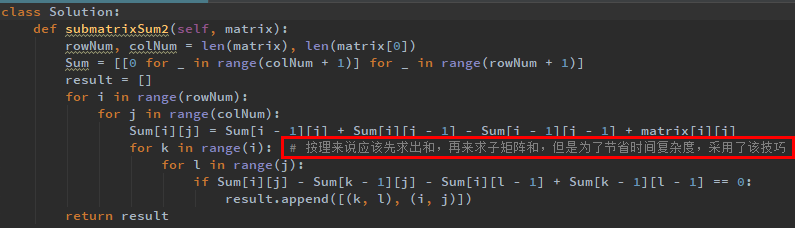
[7,8,9]]

1. 求连续矩阵和。

Sum[i][j] = Sum[i - 1][j] + Sum[i][j - 1] - Sum[i - 1][j - 1] + matrix[i][j]

1. 求子矩阵和。

Sum[i ~ m][j ~ n] = Sum[m][n] - Sum[i - 1][n] -Sum[m][j - 1] + Sum[i - 1][j - 1]



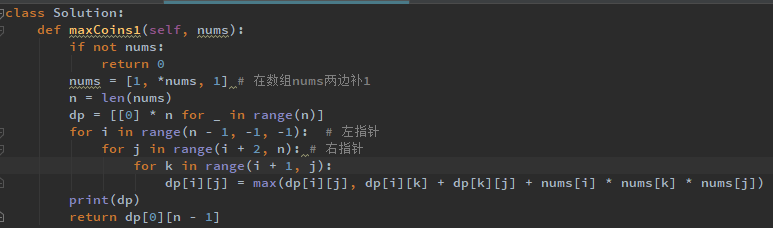
### 吹气球

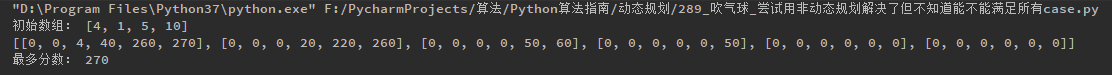
有n个气球，编号为0到n-1不等，每个气球都有一个分数，记录在nums数组中。每次吹气球i可以得到的分数为nums[left]\*nums[i]\*nums[right]，left何right分别表示与i气球左右相邻的两个气球。当i气球吹爆后欧，其左右两气球即为相邻。需要吹爆所有气球，求出能够得到的最大总得分。

我的思路：又是求最值，那肯定要用动态规划，dp[i][j]表示吹爆所有在区间[i,j]的气球，所能得到的最大分数。那么如何进行状态转移？假设吹爆第k个气球，那么k-1和k+1个气球变得相邻，不方便转移。那么我们就先吹爆区间[i,k−1]和区间[k+1,j]的所有气球，最后再来吹爆第k个气球，那么这样就十分方便转移了。枚举k的位置，得到状态转移方程为：

dp(i,j)=max{dp(i,k−1)+score(k)+dp(k+1,j)|(i≤k≤j)},

score(k)=a[l−1]∗a[k]∗a[r+1]





# 思路新颖

### 思路新颖：第k个质数

给出质数n，输出它是质数序列表中的第几个质数。

我的思路：判断一个数是否是质数好判断，但是要判断它是第几个质数就有点难办，因为要从前算到这个数，可以建立一个visit数组，如果该位置是质数就判1。

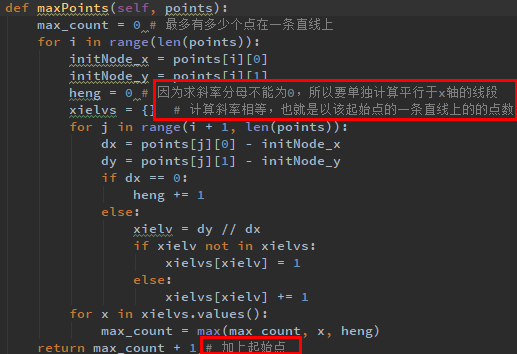


### 思路新颖：最多有多少个点在一条直线上

给出二维平面上的n个点，求最多有多少个点在同一条直线上。

我的思路：得求出一元一次方程，但是那样的话就会有多个一元一次方程，再一一比较。

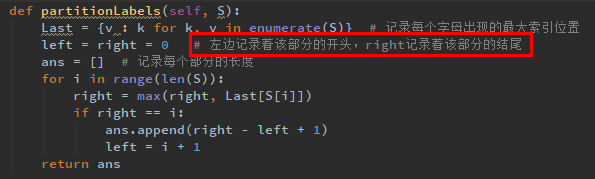
解题思路：分别以数组中的点作为起始点，其后的点相对于起始点求斜率，斜率相等说明在同一条直线上，计数不同斜率的值的数目，取最大值，然后换一个起始点反复计算。这里需要注意斜率不能为0，所以平行于x轴的线段需要另算。



### 思路新颖：分割标签

给出一个由小写字母组成的字符串，将这个字符串分割成尽可能多的部分，使得每个字母最多只出现分割后的一个部分中，并且返回每部分的长度。

解题思路：用两个指针，left指向切割字符串的开头，right指向切割字符串的结尾，right用于指向当前经过字母的那个最大索引，当right == 最大索引时，切割字符串。



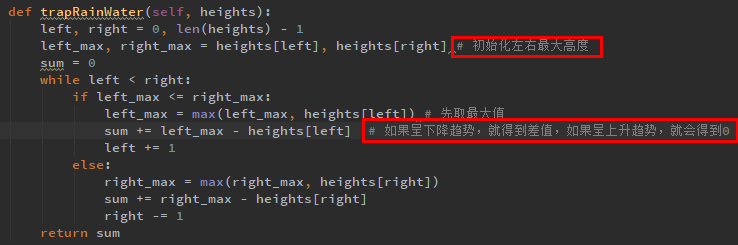




### 思路新颖：接雨水

给出n个非负整数，代表海拔图中每个宽度为1的区域的海拔，计算这个海拔图最多能接住多少（面积）雨水。

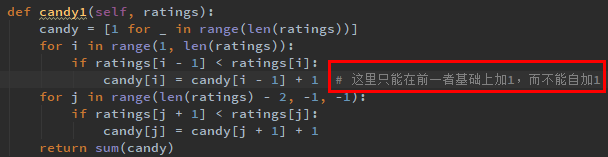
解题思路：从左边和从右边找出呈下降趋势的的地方，然后记录其高度差。低高度的指针往前移动，因为要保证下降趋势不能直到0，直到左右指针相等，完成计算。



### 思路新颖：分糖果

有N个小孩站成一列，每个小孩有一个评级。按照要求给小孩分糖果：1、每个小孩至少得到一颗糖果；2、评级越高的小孩比他相邻的两个小孩得到的糖越多。求出最少需要准备的糖果数。

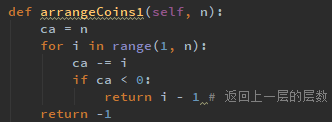
我的思路：初始的时候都是都先给每个小孩一颗糖，然后从左往右遍历，如果右边的小孩比左边的评级高，则给右边的小孩在左边小孩的基础上加一颗糖，反之从右往左遍历亦然。



### 思路新颖：硬币摆放

有n枚硬币，想要摆放成阶梯形状，即第k行恰好有k枚硬币。给出n，找到可以形成的完整楼梯行数，n是一个非负整数，且在32位有符号整数范围内。

我的思路：循环减，不满足条件时跳出。



解题思路：根据等差数列求和， n = (1 + x) \* x / 2, 求解x的值，并向下取值。



### 思路新颖：硬币排成线1

有n个硬币排成一条线，两个参赛者轮流从右边依次拿走1或2个硬币，直到没有硬币为止，拿到最后一枚硬币的人获胜，判断第一个玩家是输还是赢，若第一个玩家赢则返回True，否则返回False。

我的思路：当硬币数量是3的倍数的时候，后拿的赢，否则先拿的赢。



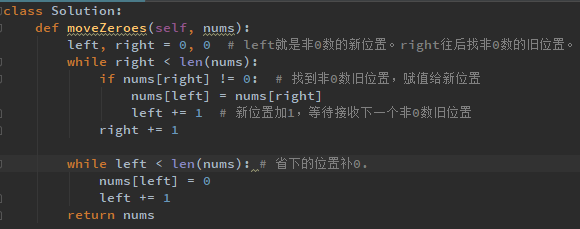
### 思路新颖：移动0问题

给定一个数组，本例实现将0移动到数组的最后面，非0元素保持原数组的顺序。注意必须在原数组上操作，最小化操作数。

我的思路：利用冒泡排序的思想，将0排到最后。

解题思路：采用逆向思维，将所有的非0的数往前移动，直到找不到了0，然后后面还剩几个位置，那么就补几个0.

思路就是：left指向非0数新位置，right指向非0数的旧位置。





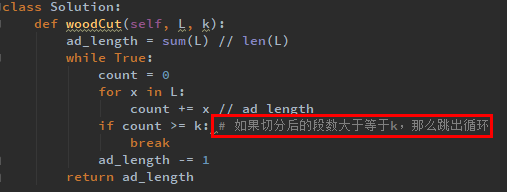
### 思路新颖：木材加工

给定一些原木，把它们分割成一些长度相同的小段木头，需要得到小段的数目至少为k，但是希望得到的小段越长越好，本例将设计能够得到小段木头的最大长度。

实例：三根木头【232、124、456】，k = 7，最大长度为114

我的思路：将n根木头相加，除以k，得到理想长度，看理想长度是否小于最小原木，然后按照理想长度去裁三根原木，看是否大于等于k，将理想长度递减，知道满足情况，就是最佳长度。

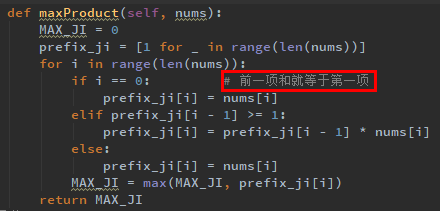
解题思路：不是按照理想长度递减，而是采用二分法，以理想长度为最大值。



### 思路新颖：乘积最大子序列

找出一个序列中乘积最大的连续子序列（至少包含一个数）

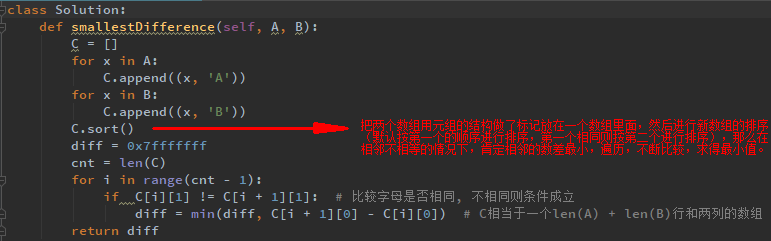
我的思路：看前n-1项积是否对第n项有积极作用，有积极作用，那么前n项积就等于前n项想乘，没有积极作用，前n项积就等于第n项。



### 思路新颖：两个整数数组的最小差

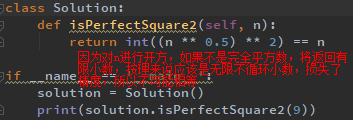
给定两个整数数组（第一个是数组A，第二个是数组B），在数组A中取A[i]，数组B中取B[j],返回A[i]和B[j]两者的最小差值A[i] - B[j]，返回最小差。

解题思路：将二者用元组做好标记后，放到同一个数组中，然后将数组排序，毫无疑问，相邻元素的差值最小，当然要保证相邻元素来自不同的数组。时间复杂度为O（2（m+n）），相比于O（m\*n）要好很多。

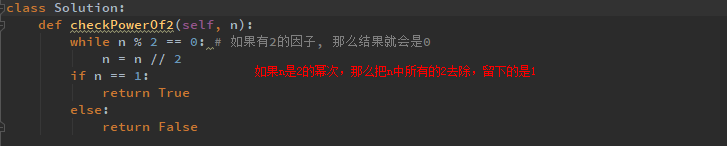


### 思路新颖：判断平方数

判断一个数是否是完全平方数

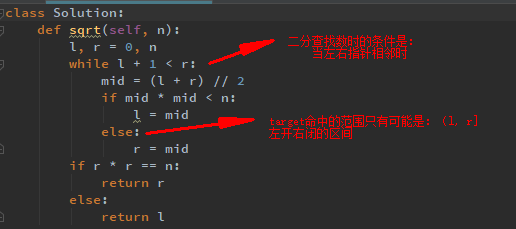


### 思路新颖：检测一个整数是否是2的倍数

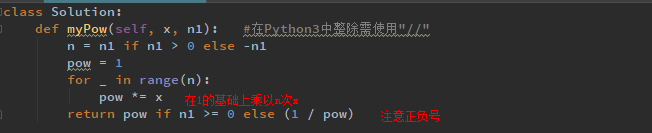


### 二分查找：求平方根

此题计算n的平方根，其实也就是求n中的最大平方的根。用二分查找相邻法，如果正好是平方数则等于r，不然则等于l。

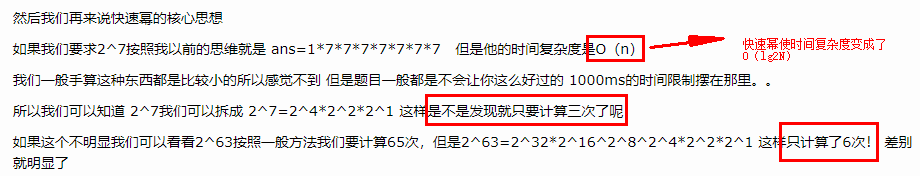


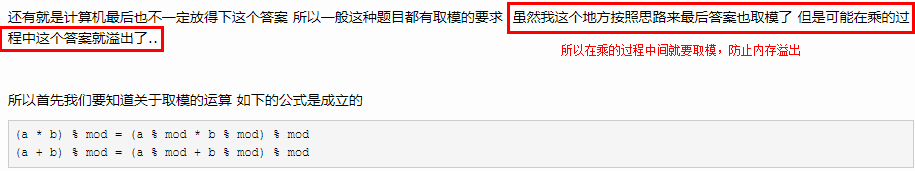
### x的n次幂

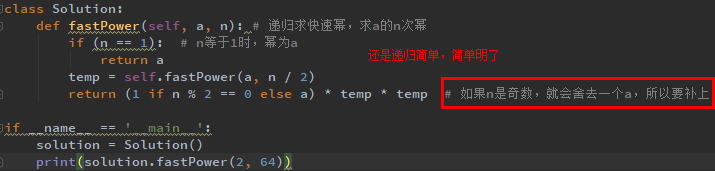


### 二分回溯：快速幂

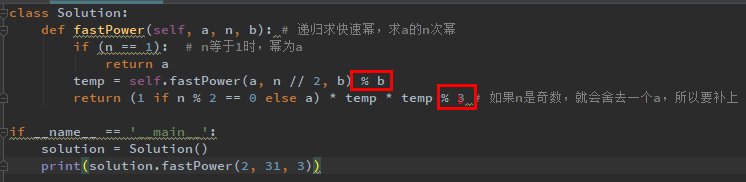
a是底数，N是指数，b是取模数，求a的n次幂，时间复杂度为O(log2N)



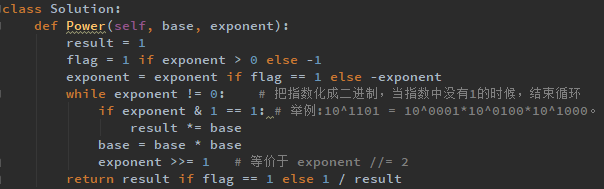




为了防止内存溢出：



快速幂之移位解法：

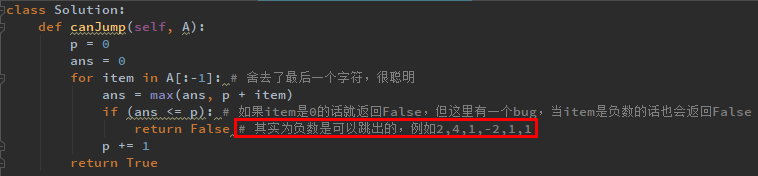


### 跳跃游戏1

给出一个非负整数数组，最初定位在数组的第一个位置。数组中的每个元素代表在那个位置可以跳跃的最大长度，本例将判断能否达到数组的最后一个位置。

我的思路：直接往后遍历就是，就是如果位置指针大于长度算到达了最后一个位置吗?

解题思路：只要不跳到0， 或者循环正负数，那么都是可以跳出的，循环正负数指2,2,1,-2,1,1



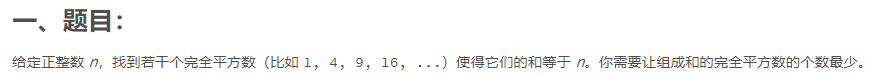
### 排颜色

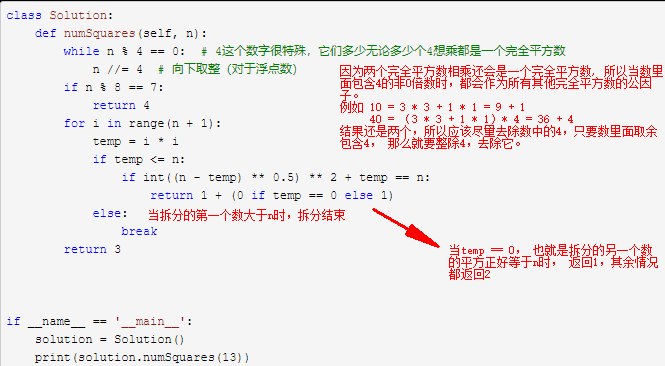
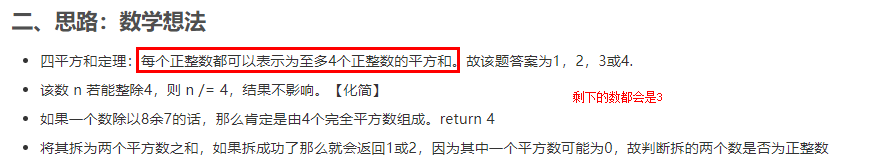
给出n个元素（包括k种不同颜色，并按照【1，k】进行编号）的数组，将其中元素进行分类，是相同颜色的元素相邻，并按照1,2....k的顺序进行排序。注意，数组应原地排序。

我的思路：其实就是含有了重复元素，相同值的元素就是相同颜色的元素。原地排序就是要求交换排序，所以该题直接使用快排就好。



### 思路新颖：完美平方



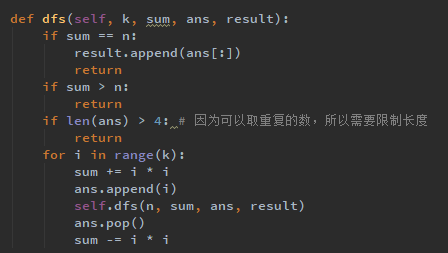


给定一个正整数n，找到若干个完全平方数（例如1,4,9），使得它们的和等于n，完全平方数的个数最少：

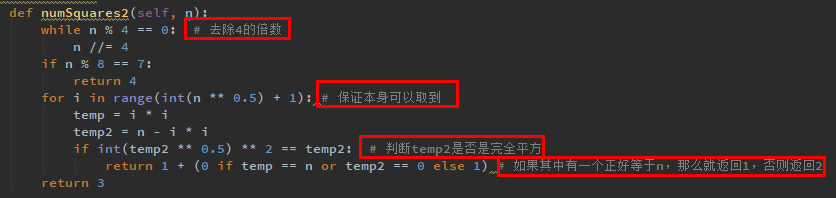
我的思路：因为一个数至多有4个完全平方数组成。

1. 去除数中的4，因为4的任何倍数都是完全平方数，无碍个数的多少
2. 如果取8的余数是7，那么就是4 (1,1,1,2)
3. 取2和取1能够判断出来
4. 省下的就是取3.

方法1：回溯



方法2：直接写



# 栈、队列

### 栈、队列：三元式解析器

给定一个表示任意嵌套三元表达式的字符串expressions，本例将计算表达式的结果。可以假设给定的表达式是有效的，并且只由数字、T、F组成（T、F分别表示True和False）。需要注意的是：

1. 给定字符串的长度
2. 每个整数都是个位数
3. 条件表达式从右到左（跟大多数语言一样）
4. 条件永远都是T或F，不会是一个数字
5. 表达式的结果总是对0-9、T或F求值。

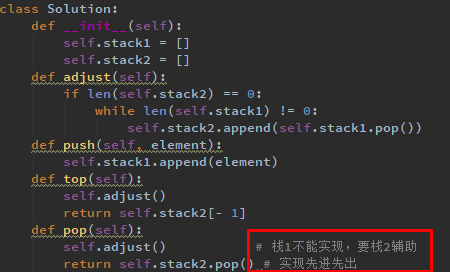
我的思路：从右往左，将数字存入栈中，栈中只存两个数，遇到T或F就选其一，存入栈中，直到栈中只剩下一个数位置。



### 栈、队列：用栈实现队列

使用两个栈实现队列的一些操作，队列应支持push、pop和top操作，其中pop可以弹出队列中的第一个（最前面）元素，pop和top方法都返回第一个元素的值

我的思路：用一个栈实现顺向，用另一个栈实现反向。



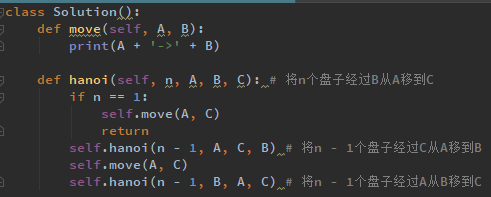
### 栈、队列：用栈模拟汉诺塔问题

在经典的汉诺塔问题中，有3个塔和N个可用来堆砌成塔、不同大小的盘子。要求盘子必须按照从小到大的顺序从上往下堆（即任意一个盘子必须堆在比它的盘子上面）。同时，必须满足三个条件：

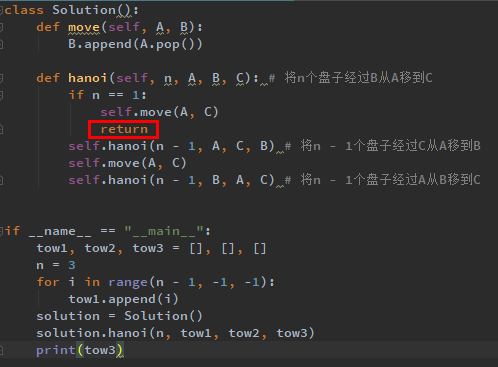
1. 每次只能移动一个盘子
2. 每次盘子从堆的顶部被移动后，只能置放于下堆中
3. 每个盘子只能放在比它大的盘子上面。

本例写一段程序，将第一堆的盘子移动到最后一堆中。

普通汉诺塔问题：



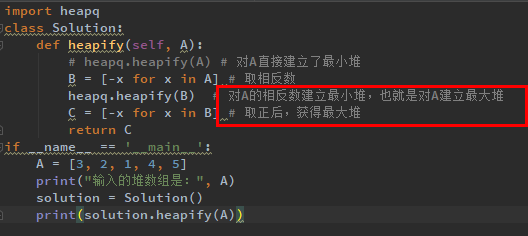
当n == 3，A，B，C = a, b, c



，只要改变move即可。

### 栈、队列：堆化操作

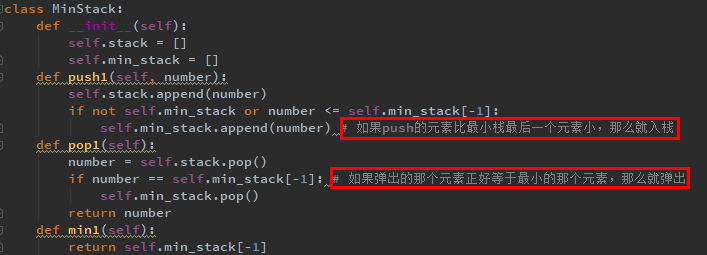
对数组建立最大堆



### 栈、队列：带最小值操作的栈

本例将实现一个带有取最小值min方法的栈，min方法将返回当前栈中的最小值，实现的栈可以支持push、pop和min操作，如果堆栈中没有数字则不能进行min方法的调用。

我的思路：很明显是要建立一个栈类，栈类中有上述三个方法push、pop和min



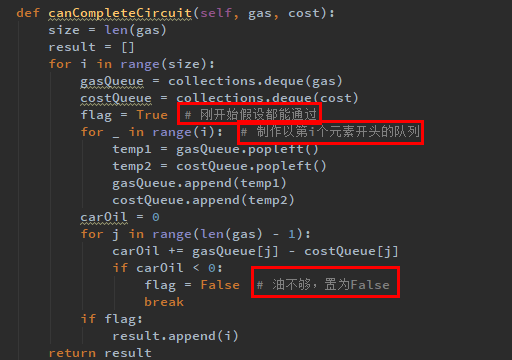
### 栈、队列：加油站

在一条环路上有N个加油站，其中第i个加油站有汽油gas[i]，并且从第i个加油站前往第i + 1个加油站需要消耗汽油cost[i]。

有一辆油箱容量无限大的汽车，现在要从某一个加油站出发环绕环路一周，一开始油箱为空。求出可以环绕环路一周时出发的加油站编号，若不存在可行方案，则返回-1.

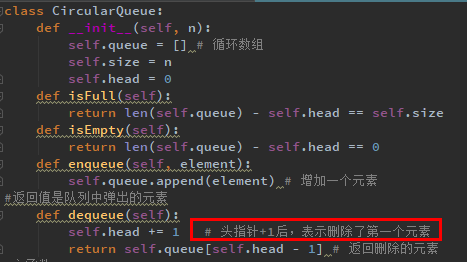
示例：现在有4个加油站，汽油量gas[i] = [1,1,3,1]，环路旅行时消耗的汽油量cost[i] = [2,2,1,1]，则出发的加油站编号为2.

我的思路：用queue制作以第i个元素开头的队列。



### 栈、队列：用循环数组来实现队列

本例将用循环数组实现队列



### 栈、队列：滑动窗口内唯一元素数量和

给定一个数组和一个滑动窗口的大小，求每一个窗口内唯一元素的个数和。注意，当滑动窗口的大小大于数组长度时，可以认为窗口大小就是数组长度，即窗口不会滑动。



### 栈、队列：逆波兰表达式求值

求逆波兰表达式的值，在逆波兰表达式中，有效的运算符号包括+、-、\*、/，每个对象既可以是整数，也可以是另一个逆波兰表达式。

我的思路：肯定需要用到栈，跟一般的还不同。

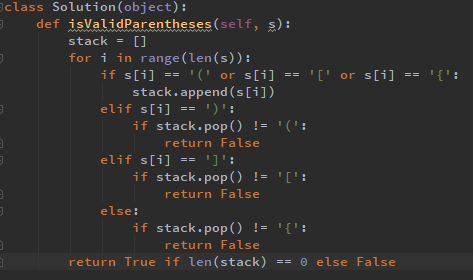
解题思路：当遇到数字时，往栈里面存入，当遇到运算符号时，弹出两个数字，进行计算，计算后的到的值再存入栈中，循环往复，知道栈中只剩下一个数，就是值。



### 栈、队列：有效的括号序列

给定一个字符串所表示的括号序列，包含小括号，中括号和大括号，各种括号要相互配对，问该字符串是否正确配对。

我的思路：建立一个栈，当遇到左括号（小、中、大）就入栈，当遇到右括号就弹栈，看弹栈出来的左括号是否和右括号匹配。

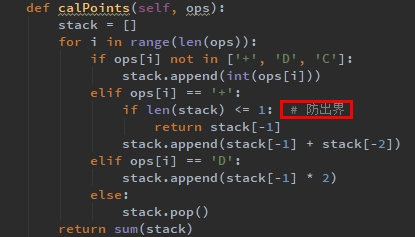


### 栈、队列：棒球游戏

对于棒球比赛成绩记录，给定一个字符串数组，每一个字符串可以是以下4种中的其中一种：

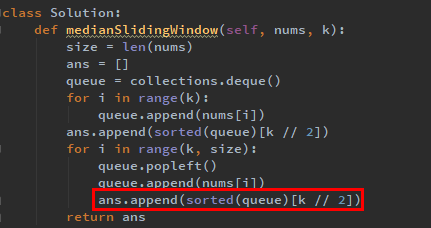
1. 整数，直接表示这个回合所得分数
2. ‘+’，表叔这个回合的分数是前两个有效分数之和
3. ‘D’，表示这个回合得到的分数是上一次获得的有效分数的两倍
4. ‘C’，表示上个回合的有效分数是无效的，需要移除。

我的思路：像这种涉及到符号和整数的，一般而言，都要用到栈，用栈存入数字，遇到操作符，就根据操作符的操作进行操作。



### 栈、队列：滑动窗口的中位数

给定一个包含n个整数的数组和一个大小为k的滑动窗口，从左到右在数组中滑动这个窗口，找到数组中每个窗口内的中位数。注意，如果数组格式是偶数，则在该窗口排序数字后，返回第N//2个数字。



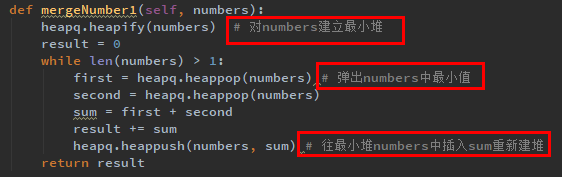
### 栈、队列：合并数字

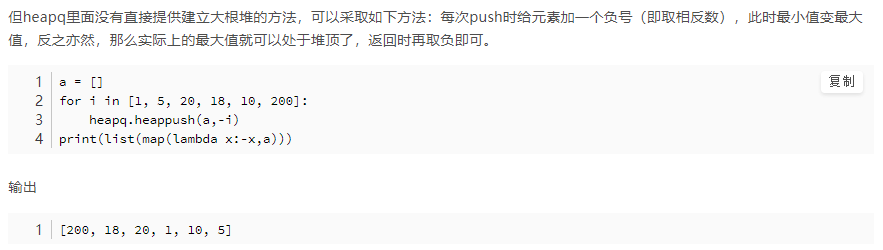
给定n个数，将这n个数合并成一个数，每次只能选择两个数a、b合并，合并需要消耗的能量为a+b，输出将这n个数合并成一个数后消耗的最小能量。

我的思路：明显每次挑选最小的两个数求和，然后再将和放回到列表中，再调出两个最小的值求和，再放回列表，循环反复，当只剩一个数时，就是n个数合并的最小能量。

明显即使是最小能量也会大于所有数的和，因为会进行重复的加法。

可以利用heapq结构，取出最小的两个数，将和加入heapq进行自动堆排序。

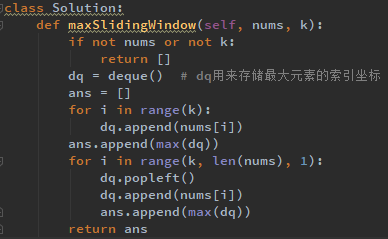




### 栈、队列：滑动窗口的最大值

给定一个可能包含重复整数的数组和一个大小为k的滑动窗口，从左到右在数组中滑动这个窗口，找到数组中每个窗口内的最大值。

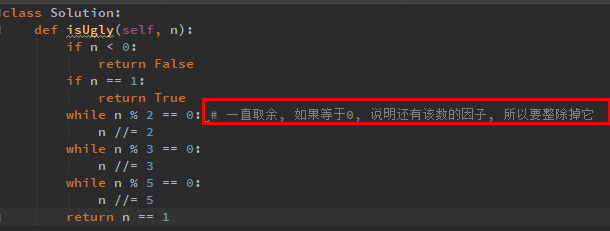
我的思路：用queue初始化窗口为k，然后从左边剔除一个元素，从右边加入一个元素，然后比较最大值。



### 栈、队列：丑数1

丑数的定义是：只包含质因子2、3、5的正整数，例如6、8就是丑数，但14就不是丑数，因为它包含质因子7，本例将检测一个整数是不是丑数。

我的思路：将整数里的所有2的倍数、3的倍数、5的倍数都去掉，如果剩下1就说明是丑数。注意，1、2、3、5都是丑数。

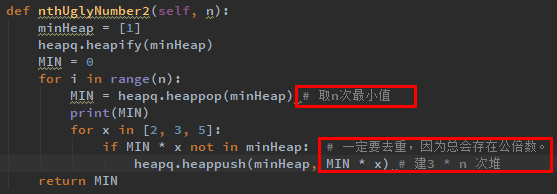


### 栈、队列：丑数2

设计一个算法，找出只含素因子2、3、5的第n小的数，符合条件的数如：1、2、3、4、5、6、8、9、10、12.....

我的思路：按顺序找，但是效率低下。

解题思路：从1开始，存入heapq，依次乘【2、3、5】，都存入heapq，然后再取出最小值，再依次成【2、3、5】，当取了n次时，就是第n小的数。



### 栈、队列：超级丑数

超级丑数的定义是：所有质数因子都是给定一个大小为k的质数集合的正整数，例如，给出4个质数的集合【2,、7、13、19】，那么【1、2、4、7、8、13、14、16、19、26、28、32】是前12个超级丑数，本例将找出第n个超级丑数。

我的思路：上题的再运用。

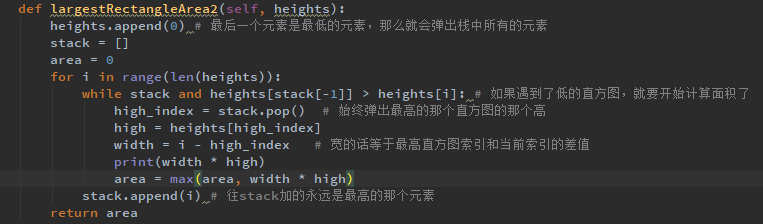


### 栈、队列：直方图中最大的矩阵面积

给出n个非负数表示每个直方图的高度，每个直方图的宽均为1，在直方图中找到最大的矩形面积。

我的思路：首先明确要用到栈，存入栈里面的是直方图的索引，因为既可以通过索引找到高，又可以通过索引确定宽。

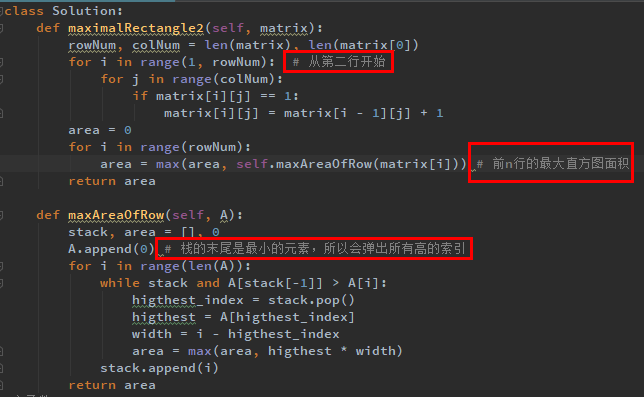
只要遇到比栈中最后一个元素高的直方图就将其索引存入栈中，遇到低的就弹栈，然后计算宽，比较最大面积。



### 栈、队列：最大矩形

给定一个二维矩阵，元素取值0和1，找到一个最大的矩形，使得其中的值全部为1，输出它的面积。

我的思路：将数组给矩形化，当数组元素为0时，那就是0；当数组元素是1时，那么就等于上一行元素的值加上该行的1的和，把前n行当做求最大直方图的最大矩形面积。

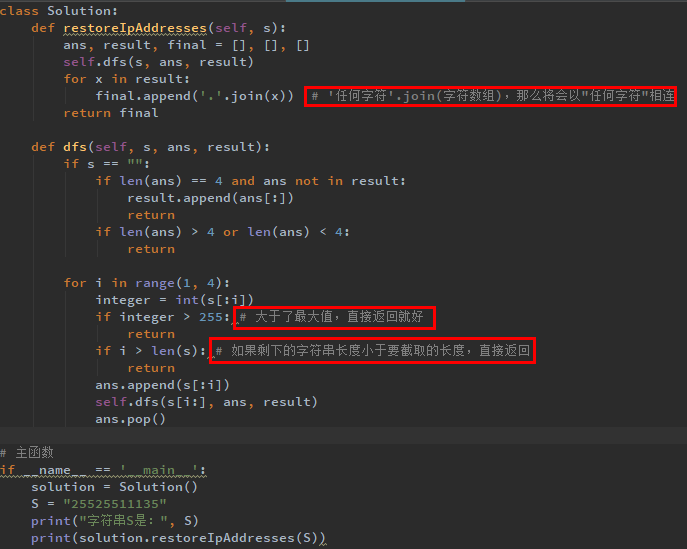


# 回溯递归

### 回溯递归：恢复IP地址

给定一个由数字组成的字符串，求出其所有可能的IP地址

我的思路：毫无疑问该用回溯，当字符串被切割成空时， 如果分割的部分小于4或大于4，返回，并且长度为4时，记录并返回。

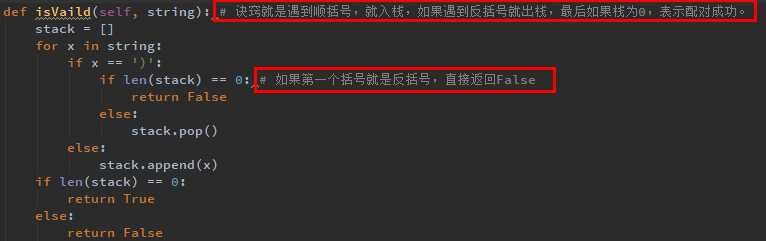


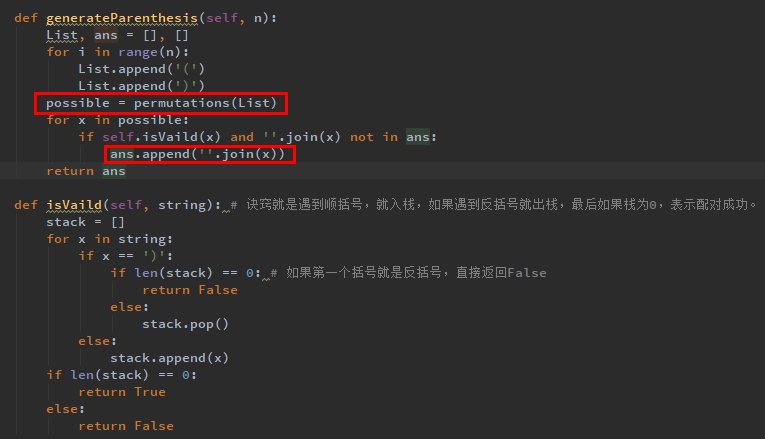


### 回溯递归：生成括号

给出n对括号，将这些括号任意组合，生成新的括号组合，并返回所有可能组合结果

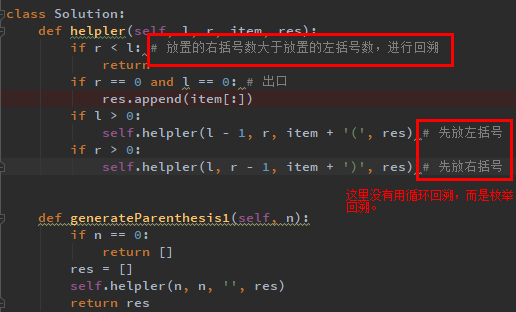
我的思路：利用permutation函数对其进行全排列，看满足条件的有多少种。其中判断满足条件的函数如下：





解题思路：回溯先放左括号还是先放右括号

1. 左括号和右括号最多可放n个
2. 当放置的左括号数小于右括号数，不可放置右括号。



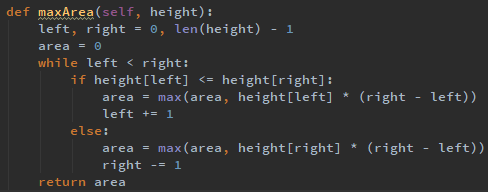
### 回溯递归：装最多水的容器

给定n个非负整数a1，a2,....,an，每个数代表坐标中的一个点（i，ai），画n条垂直线，使得垂直线i的两个端点分别为（i，ai）和（i，0）。找到其中两条垂直线，使其与x轴共同构成一个容器，以容纳更多水。

我的思路：感觉就是求最大直方图面积，但是最大直方图其实提供的是每次提供的是两条垂直线，而这里只有一条。

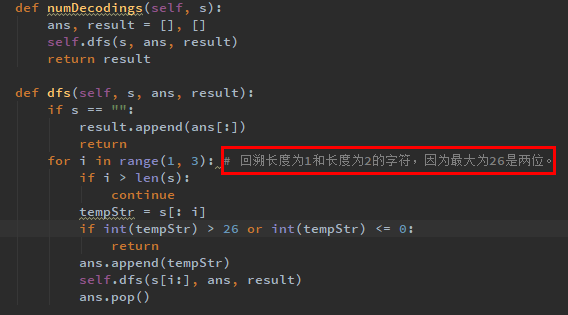
解题思路：两种情况可能取得最大值，足够宽和足够高。

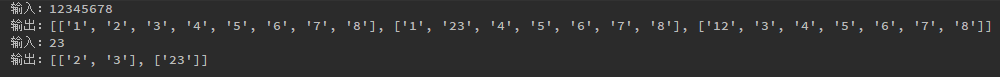
从左边找一条高，从右边找一条高，以短的那条为高，两者的距离为宽，计算面积，短的那边往中间移动，找更高的边长，继续计算，循环比较，得出最大面积。



### 回溯递归：解码方法

有一个消息包含A-Z,并通过以下规则编码，A：1，B：2，....，Z：26.现在给定一个加密后的消息，求出所有编码方式的数量。



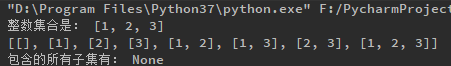
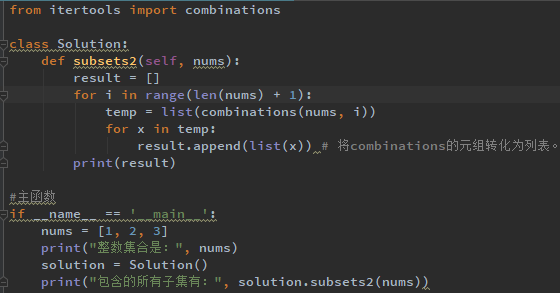


### 回溯递归：子集2

给定一个含不同整数的集合，返回其所有的子集。

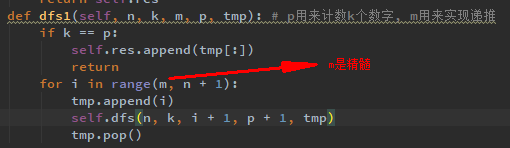
我的思路：回溯的点是整数的位置，从不同的位置取值，求出所有不同长度的子集。

但是最后是使用combinations函数进行取值。



### 回溯递归：数字组合1

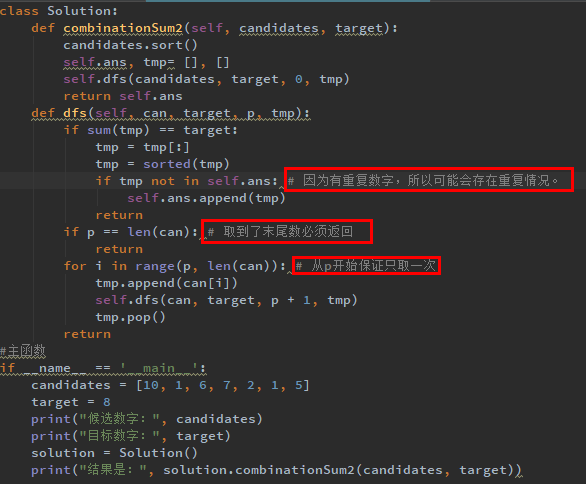
给定两个整数n和1，返回【1，n】中k个数字所有可能组合。



但毫无疑问combinations（arr，k）明显更快

### 回溯递归：数字组合2

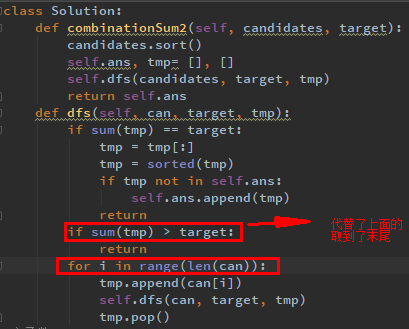
给出一组候选数字C和目标数字T，本例将找出C中所有的组合，使组合中的数字和为T，C中每个数字在每个组合中只能使用一次。





### 回溯递归：数字组合3

相对上题，改一个条件，就是C中的数字可以无限制的取用。

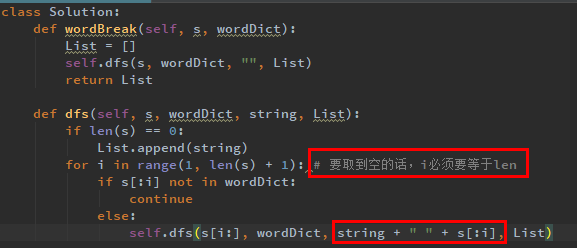




### 回溯递归：单词拆分2

给定字符串s和单词字典dict，在字符串中增加空格来构建一个句子，并且所有单词来自字典，返回所有可能的句子。

我的思路：毫无疑问要进行回溯，回溯的就是在s中并且在dict内的单词。那么出口呢就是s中的字符被完全截取了，就存取该情况，那么什么情况下返回呢



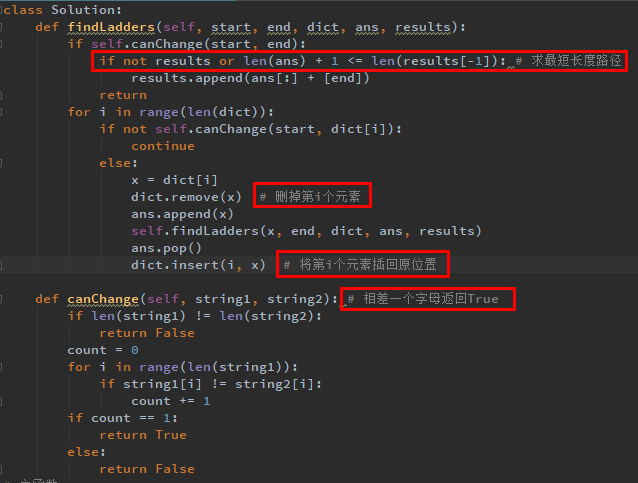
### 回溯递归：单词接龙1

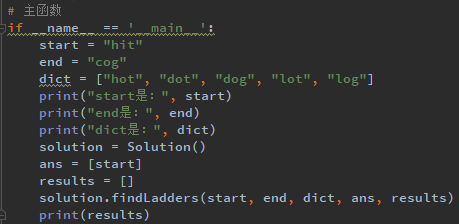
给出两个单词（start 和 end）和一个字典，找出所有从start到end的最短转换序列。转换规则为：

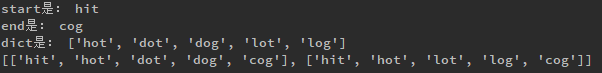
1. 每次只能改变一个字母；
2. 转换过程中的中间单词必须在字典中出现。

返回所有可能的转换情况。

我的思路：回溯数组中能够被一个单词转化的所有情况，并删除该情况，直到转化了的字符串和end只相差一个字母，记录该情况，插回删除的元素，继续回溯，知道所有情况被找出。



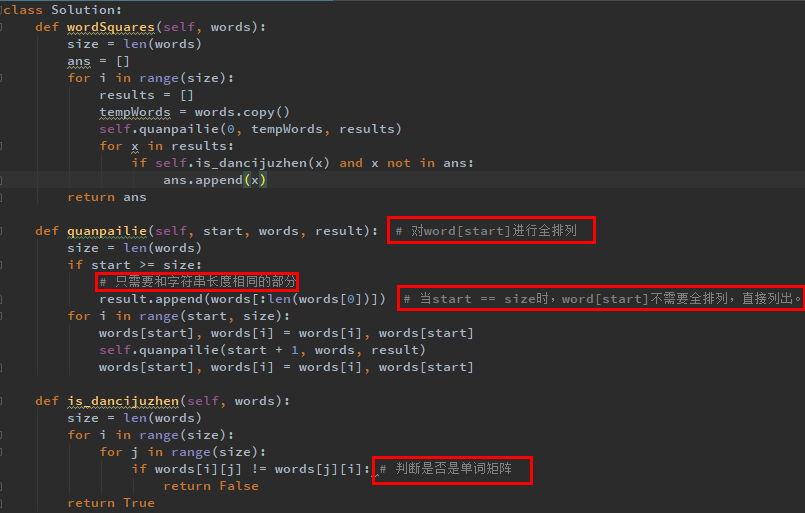




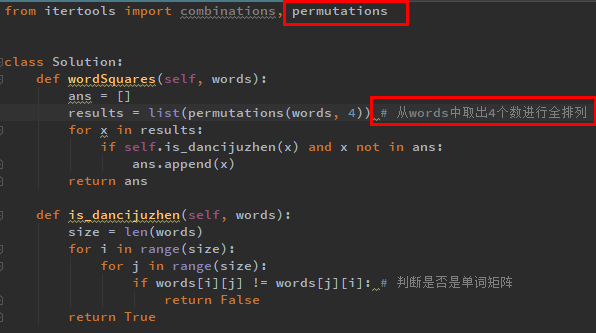
### 回溯递归：单词矩阵

给出一系列不重复的单词，本例将找出所有可能构成的单词矩阵。一个有效的单词矩阵是指，如果从第k行读出来的单词和第k列读出来的单词相同（0 <=k<max(numRows,numColumns）,那么就是一个单词矩阵。



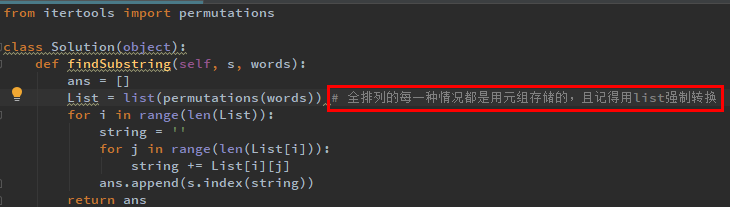


利用Python中的全排列函数进行计算。



### 回溯递归：包含所有连接的子串

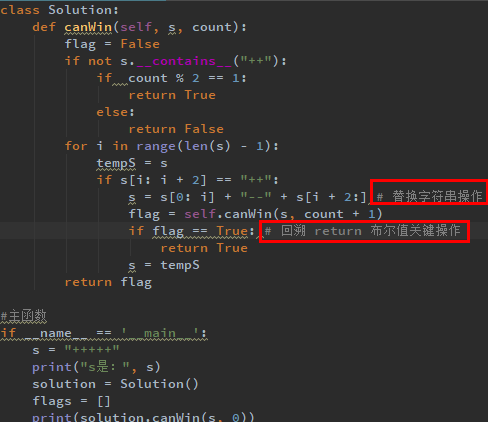
给定一个字符串s和一个单词列表words，在s中查找子串，正好是所有单词列表的连接，输出查找到子串的其实索引。

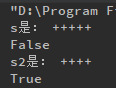


### 回溯递归：翻转游戏

给定一个只包含两种字符+和-的字符串，两个人轮流翻转“++”变成“--”。当一个人无法采取行动时游戏结束，另一个人将是赢家，本例将判断能否保证先手胜利。

我的思路：先遍历的是不同的‘++’的起始位置，然后换成‘--’后，再当做一个新字符串翻转“++”，所以回溯的是不同‘++’的位置，定义一个count，翻转一次，自加一次，当没有“++”后，如果count是奇数，说明能保证先手胜利。

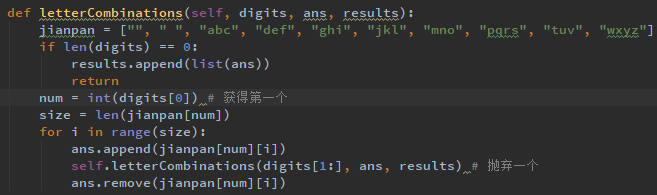




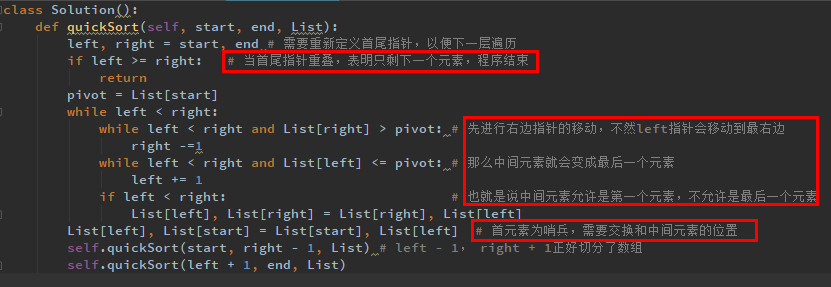
### 回溯递归：电话号码的字母组合

传统的电话拨号盘如图1所示，2-9数字键盘上的每个数字可以代表3个字母之一。例如数字2代表abc。输入任何2-9的数字组合，本例将返回所有的字母组合。

我的思路：肯定要用到回溯的方法，采用字符串截取的方法，回溯每一个字符的所有可能取值情况，出口是，当字符串被截取到空时。



### 回溯递归：整数排序



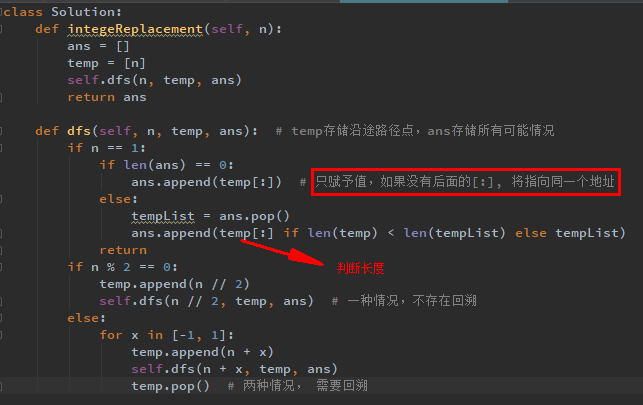


### 回溯递归：整数替换

给定一个正整数n, 如果n为偶数，将n替换为n/2, 如果n为奇数， 将n替换为n + 1 或 n - 1， 那么将n转化为1，最少的替换次数是多少？

解题思路：

这道题拿到手上肯定是一道递归的题目，函数的意义是求n转化为1，最少的替换次数是多少，那么递归就是n / 2, n + 1, n - 1转化为1，最少的替换次数是多少，而当为奇数时，有两种情况，毫无疑问要用到回溯的思想，那么出口就是，当n = 1时，把路径存起来， 那么沿途就需要存路径点，为了节省空间复杂度，出口时做出判断，此处用栈。



### 回溯递归：寻找丢失的数

给一个由1 - n的整数随机组成的一个字符串序列， 其中丢失了一个整数，本例将找到它。

比如：n = 20, str = "19201234567891011121314151618" 丢失的数是17

解题思路：该题拿到手的时候明显该使用回溯的方法，因为你无法判断你从左往右找的数字是一位还是两位，所以要根据一位两位进行回溯， 那么出口的话，一是找到了0，二是找到了重复的数字，三是到底了。那么该函数的意思就是从第几位开始往后填充存在的值。

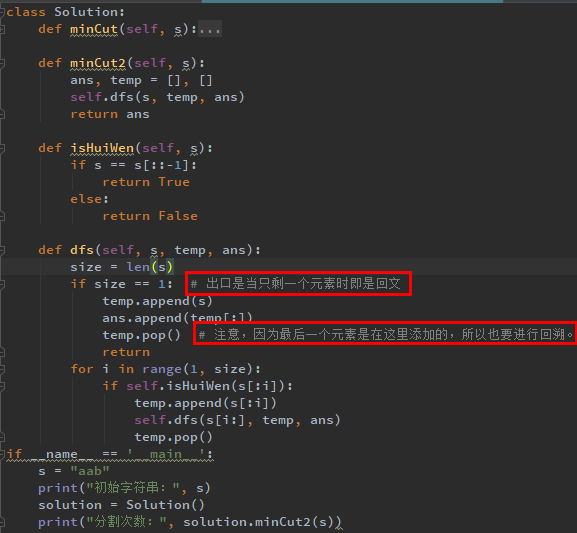


### 分割回文串1

给定一个字符串s，将s分割成一些子字符串，使每个子字符串都是回文，返回s符合要求的最少分割次数。

我的思维：肯定是能够把字符串分割成子字符串都是回文的，大不了都分割成单个的就好，就是要尽可能找到长的回文串。

其实可以用回溯的方法进行解决，找到回文长度分别为（n ~~1）的子串，知道回文长度为1是一种解法。

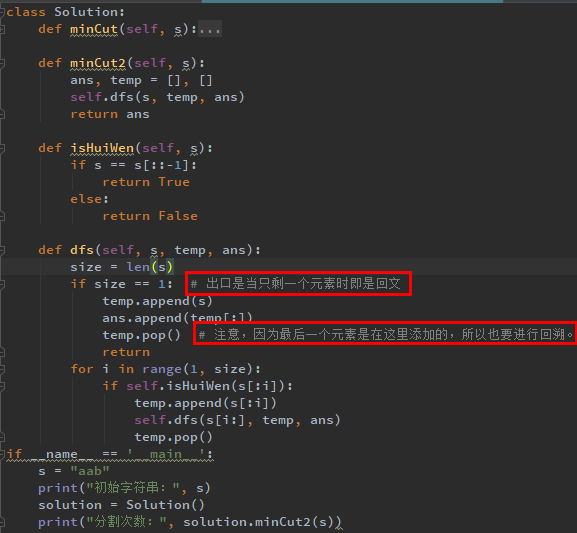


### 回溯递归：分割回文串2

给定一个字符串s，将s分割成一些子字符串，使每个子字符串都是回文串，返回s所有可能的回文串分割方案：

我的思维：肯定是能够把字符串分割成子字符串都是回文的，大不了都分割成单个的就好，就是要尽可能找到长的回文串。

其实可以用回溯的方法进行解决，找到回文长度分别为（n ~~1）的子串，知道回文长度为1是一种解法。



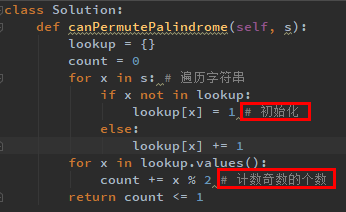
### 回溯递归：回文排列1

给定一个字符串，判断字符串是否存在回文排列，若存在则返回True，否则返回False

解题思路：其实是问用该字符串中的字符能够组成回文串。

用一个字典，键用来存储字符串中存在的字符，值用来计数字符的个数

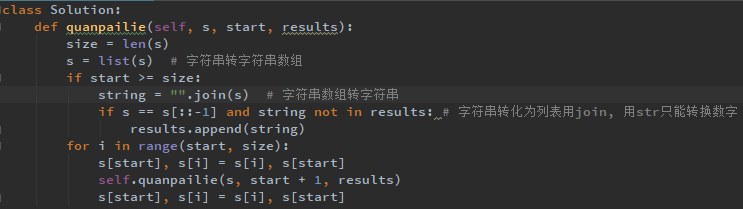
然后取出该字典的values数组，如果存在两个奇数那么就不可能组成回文 串，**只能允许有一个奇数**，放中间组成回文串。



### 回溯递归：回文排列2

给定一个字符串s,返回所有的回文排列（不重复）；如果没有回文排列，返回空列表

解题思路：用全排列方便一点



# 数据结构

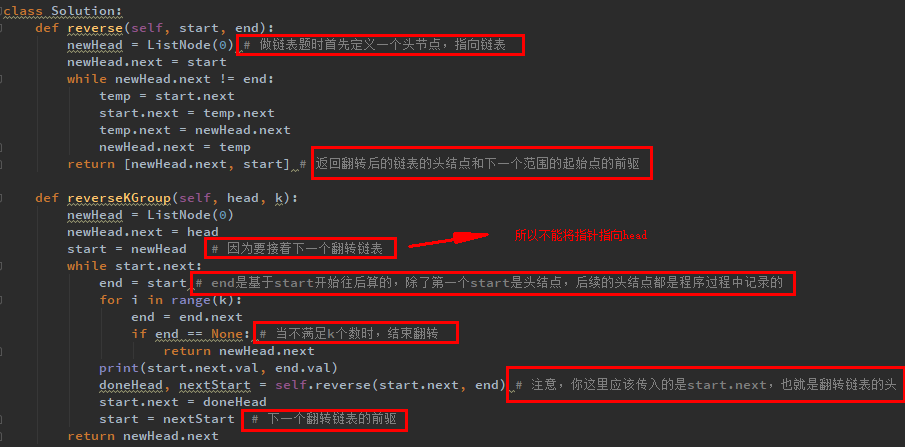
### k组翻转链表

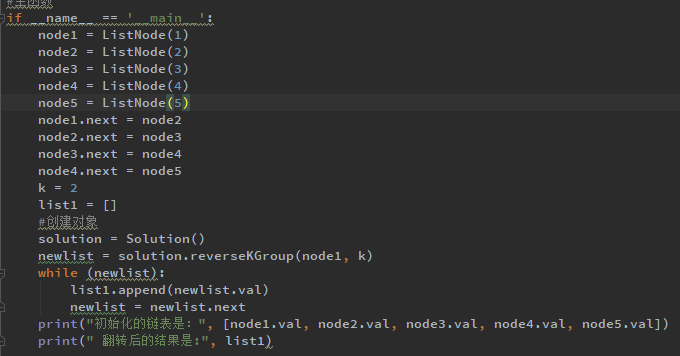
给定链表及整数k，将这个链表从头指针开始，每k个元素翻转一下。链表元素个数不是k的倍数，最后剩余的元素不用翻转。

我的思路：首先需要两个函数：

1. 翻转指定范围的链表：reverse（start，end）；其中start和end分别是起始节点和结束节点，返回的是翻转后链表的头节点和下一个翻转链表的开头。
2. 找到整个链表的各个范围的头结点和尾节点

其中reverse链表利用的是start指针一直往后移，而start经过的节点都放到头节点，等start到了end后，自然而然就完成了翻转了。







### 线段树的修改

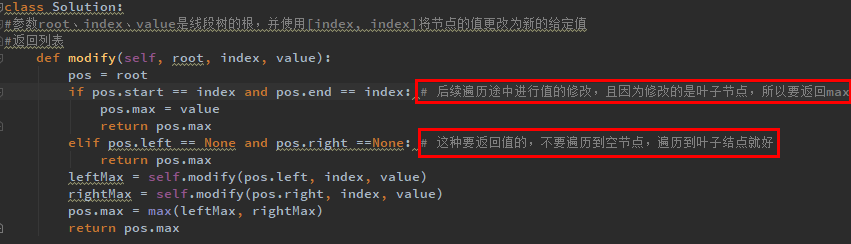
修改线段树的末尾节点，确保修改后，线段树的每个节点max属性仍然具有正确的值。

我的思路：查找二叉树，找到该点后，进行修改，然后对max重新构造树。

如何重新构造树？

采用后续遍历的方法，根节点取左右子树的更大值，return根节点。

解题思路：不用将查找和修改分开，后续遍历途中就可以进行值的更改。



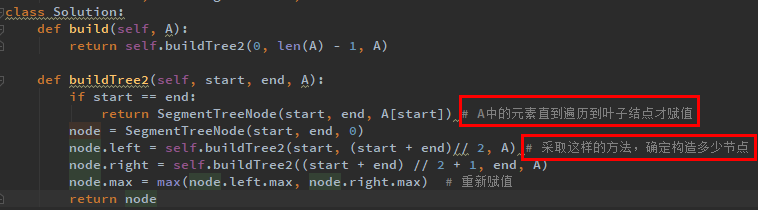
### 线段树的构造1

根据列表构造线段树，线段树的叶子节点的start==end==列表索引，max=列表中的值

我的思路：毫无疑问应该采用后序遍历进行树的构造。

解题思路：上述思路错误，应该采取前序遍历构造，先构造父节点，然后构造子节点，直到列表中的元素不再支持构造子节点，返回构造好的叶子节点。

因为这里构造的是线段树，所有叶子结点加起来就是列表中的所有元素，所以这里构造很特殊，非叶子节点的max其实都是0，等到叶子节点构造完成后，在重新更改值，列表中的值在遍历到叶子节点是赋值。



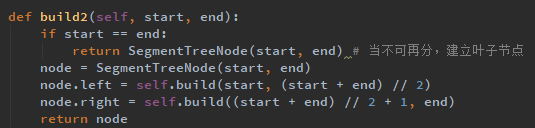
如果是构造普通的二叉树

### 线段树的构造2

本例将实现一个build方法，接收start和end作为参数，然后构造一个代表区间【start， end】的线段树，返回这棵线段树的根。这棵线段树不是叶节点全是列表中的点，而是将区间中的数不断二分，分到为一个数作为叶子节点位置。

我的思路：肯定还是用前序遍历，先建立父节点，再建立子节点。

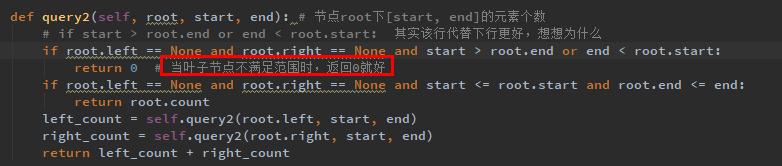
相比于上一题其实就是少了一个max。



### 线段树查询1

找到线段树中指定区间【start，end】内的元素个数

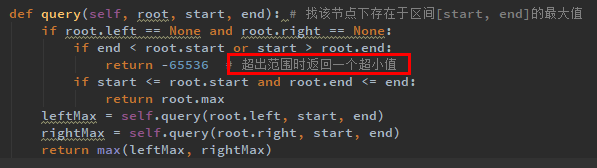
我的思路：其实元素总个数就是叶子节点的count值相加，那么其实只要遍历到叶子节点，看该节点的范围是否符合【start， end】，如果符合，那么相加就好了。



### 线段树查询2

跟上题相比，找的不是区间【start，end】内的值之和，而是区间内的最大值。

我的思路：其实就是遍历到叶子节点就直接return max，而后记录left\_max,right\_max，最后返回其中更大值，一直返回到根节点。



### 是否为子树

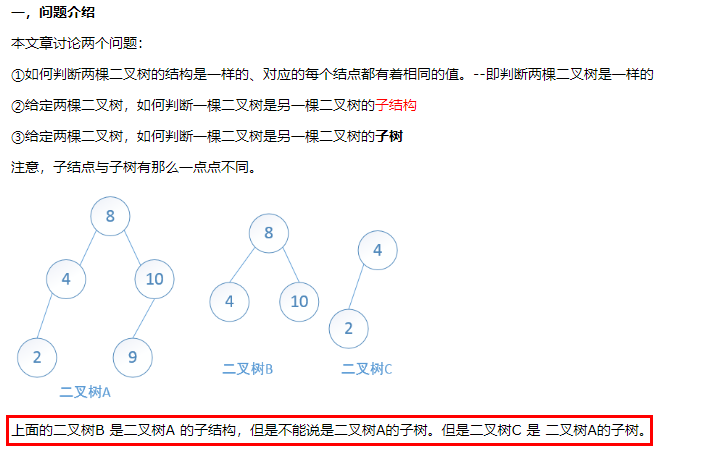
有两个大小不同的二叉树，判断一个是否是另一棵的子树：

我的思路：用同一种遍历，求出二者的遍历序列，看其中是否存在一者包含另一者的情况。

解题思路：我的思路是错误的，因为不同结构的树也可能遍历序列相同。

由于先序遍历（后序遍历） 再加上中序遍历能唯一确定一棵二叉树（必须知道中序遍历，因为只有中序遍历才能知道根节点的左右各有多少个节点）

**拓展：**

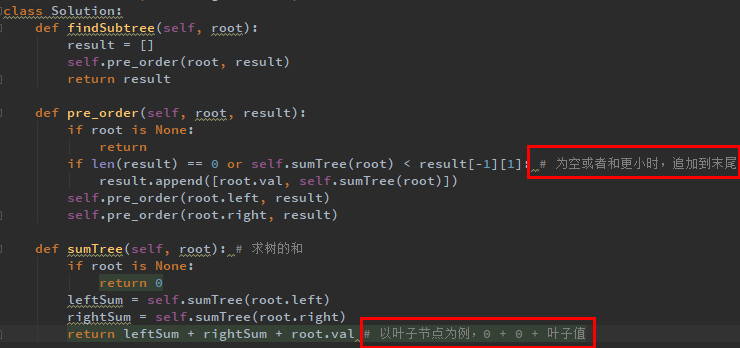


**二叉树的子树和子结构  
子树的意思是只要包含了一个结点，就得包含这个结点下的所有节点.  
子结构的意思是包含了一个结点，可以只取左子树或者右子树，或者都不取。**

### 最小子树

给定一棵二叉树，找到和为最小的子树，返回其根节点：

我的思路：1、定义一个求树和的函数 2、遍历树，记录最小和及其根节点



### 具有最大平均数的子树

给定一棵二叉树，找到具有最大平均值的子树，返回子树的根节点，并输出子树

我的思路：1、定义一个函数求树的平均值 2、遍历该树，比较最大值。



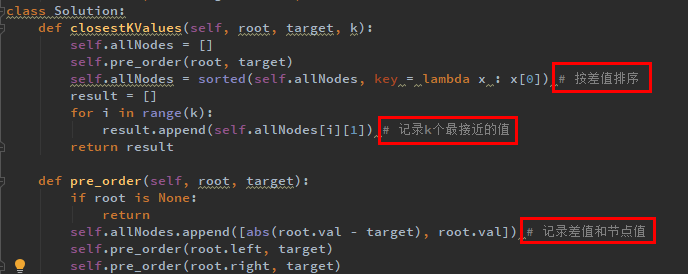
### 二叉搜索树中最接近的值

给定一棵非空二叉搜索树（BST）以及一个target值，找到其中最接近给定值的k个数。注意，1、给出的targer值是浮点数；2、可以假设k总是合理的，即k小于等于总结点数；

1. 可以保证在给出BST中只有唯一一个最接近给定值的k个值的集合。

我的思路：利用中序遍历序列化得到排序的数组，然后按照找到target，从中间往两边发散找到k个值。

解题思路：思路误区，只是要你找出k个最接近target的数，并没有说要连续。所以只需要遍历树，记录树节点值和target的差和树节点，按差从小到大排序取k个数。

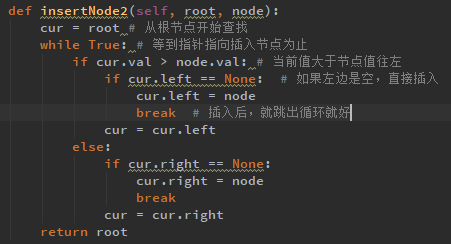


### 二叉搜索树中插入节点

给定一棵二叉搜索树和一个新的树节点，将节点插入树中，需要保证该树仍然是一颗二叉搜索树。

我的思路：无

解题思路：陷入思维误区，往二叉搜索树中插入节点，肯定是作为叶子节点插入的，所以不需要遍历，直接循环就好，大于节点值往右，小于节点值往左，如果向左向右为空了，那就直接插入，跳出循环，返回根节点。



### 二叉搜索树中删除节点

给定一棵具有不同节点值的二叉搜索树，删除树中与给定值相同的节点。如果没有，就不做任何处理，如果有，保证处理之后树仍然是一棵二叉搜索树。

我的思路：利用循环找到要删除节点，将节点替换为该节点的右节点，然后将该节点的左节点变成右节点的左节点。

解题思路：记录非删除节点，对非删除节点构造树



### 二叉搜索树转化成更大的树

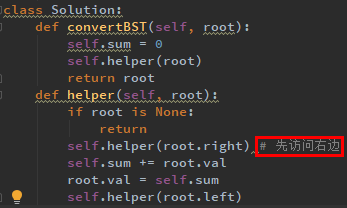
给定二叉搜索树，将其转换为更大的树，使原始BST上每个节点的值都更改为在原始数中大于等于该节点值之和（包括该节点）

我的思路：没理解题目意思

解题思路：就是每个节点值应该等于大于该节点值的其他节点值之和加上该节点的值

变相使用中序遍历，中序遍历是左中右，改为右中左的顺序。进行遍历，传递一个sum值，将经过的sum值加到经过的节点的val中。

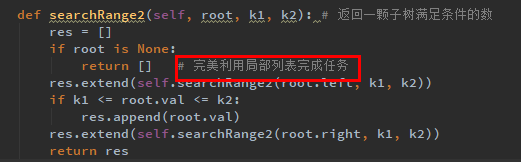
因为比根节点大的就是其右子树，比左节点大的是根节点和兄弟右子树。



### 二叉搜索树的搜索区间

给定两个值k1、k2（k1 < k2）和一个二叉搜索树的根节点。找到树中所有值在k1到k2范围内的节点，并打印所有x，其中x是二叉搜索树的节点值，返回所有升序的节点值。

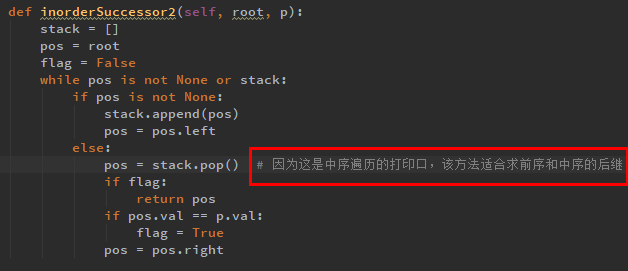
我的思路：中序遍历所有节点，记录在范围内的值



### 二叉搜索树的中序后继

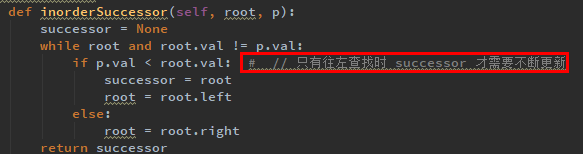
给定一棵二叉搜索树以及一个节点，采用中序遍历的方法，求给定后继节点，如果没有后继节点则返回null。

我的思路：利用中序遍历将二叉搜索树序列化。



从中序遍历的特性去寻找：左-根-右。中序遍历一个结点时，下一个结点有三种情况：

1. 如果当前结点有右结点，则下一个遍历的是右子树的最左结点；
2. 如果当前结点无右结点，若它是父节点的左儿子，则下一遍历的是父节点；
3. 如果当前结点无右结点，且它是父节点的右儿子，则所在子树遍历完了。向上寻找一个作为左儿子的祖先结点，那么下一遍历的就是该祖先结点的父节点；（一直找到根节点为止）
4. 如果上面三种情况都没找到，则该节点是树的最后一个结点，无后继结点

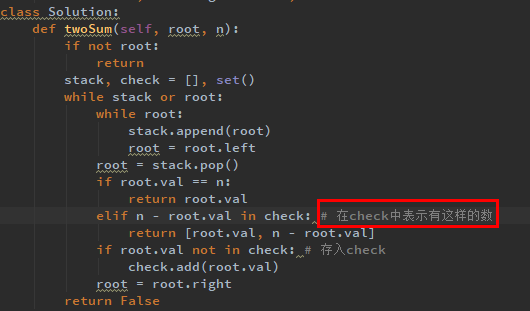


### 二叉搜索树两数之和

给定一个整数n，在二叉搜索树中找到和为n的两个数字返回

我的思路：这种情况是不是有很多种？岂不是需要回溯

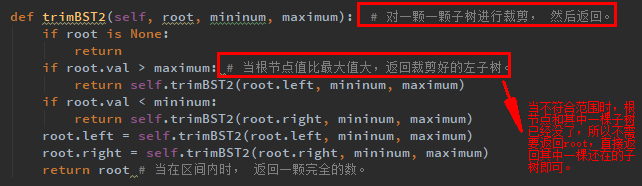
该题代码有问题，只能返回一种情况



### 裁剪二叉搜索树

给定一棵有根的二叉搜索树和两个数字min、max，将这个数超出范围的数裁掉，依然保证该数是一颗合法的二叉搜索树。

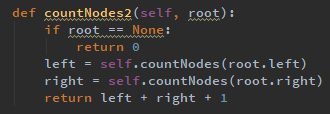
我的思路：肯定利用构建二叉树的思想，进行前序遍历，然后看是否符合范围，不符合范围，看大于还是小于，如果大于，那么舍弃掉根节点和右子树，反之亦然。直到为None



### 统计完全二叉树节点树

给定一棵完全二叉树，计算它的节点数

我的思路：为空时返回0，之后返回左子树的节点数+右子树的节点数+1

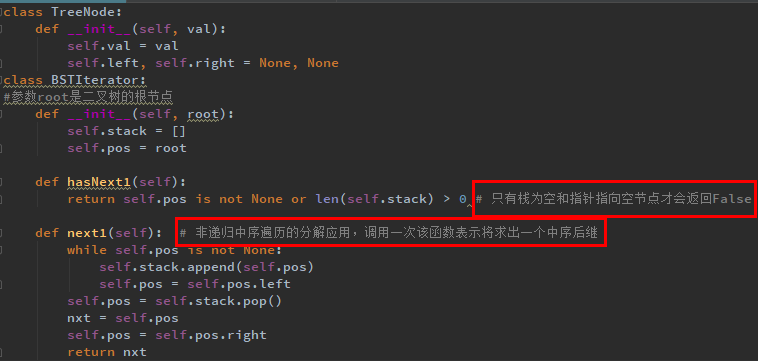


### 二叉搜索树迭代器

本例将实现一个带有下列属性的二叉查找树迭代器：

1. next（）返回二叉树中下一个最小的元素；
2. 元素按照递增的顺序被访问（例如中序遍历）
3. Next（）和hasNext（）的询问操作要求平均时间复杂度为O（1）

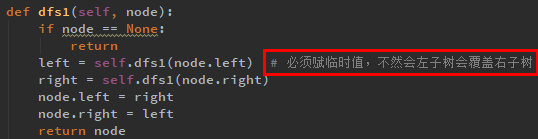
我的思路：如何利用中序遍历实现时间复杂度为O（1）？



### 翻转二叉树

翻转一棵二叉树，即所有非叶子节点的左右子树交换。

我的思路：从根节点开始，重新构造树，左右节点互换。

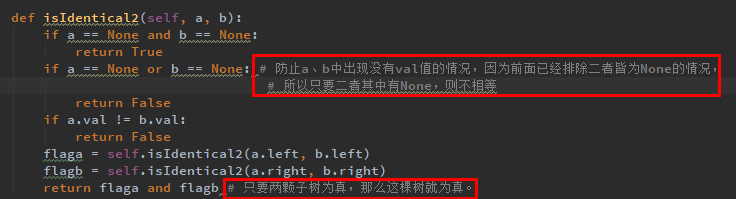


### 相同二叉树

检查两颗二叉树是否等价，即两颗二叉树必须拥有相同的结构，并且每个对应位置节点上的数都相等。

我的思路：因为前序遍历（后序遍历）和中序遍历能唯一确定一棵树。

解题思路：两棵树同时从根节点开始遍历，值不相等就返回False，或者其中一个为None另一个不为None，当然同时为None也可以返回True，接下来再比较两个子树是否相等，最后返回两颗子树的布尔值相与，都为真是为真。

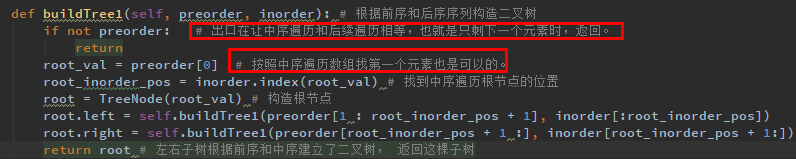


### 前序遍历树和中序遍历树构造二叉树

根据前序遍历和中序遍历的结果构造二叉树

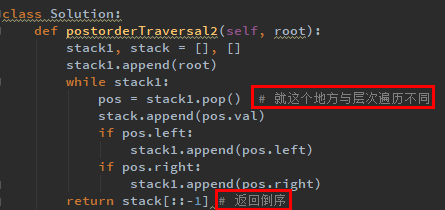
我的思路：其实就是要把前序遍历和中序遍历的相同的部分分在一起，知道只剩一个元素时，那么前序遍历和中序遍历是相等的。那么这就是出口。

解题思路：构造二叉树肯定是前序遍历的，先构造根节点，在构造两个子节点，最后返回根节点。



### 二叉树的后续遍历

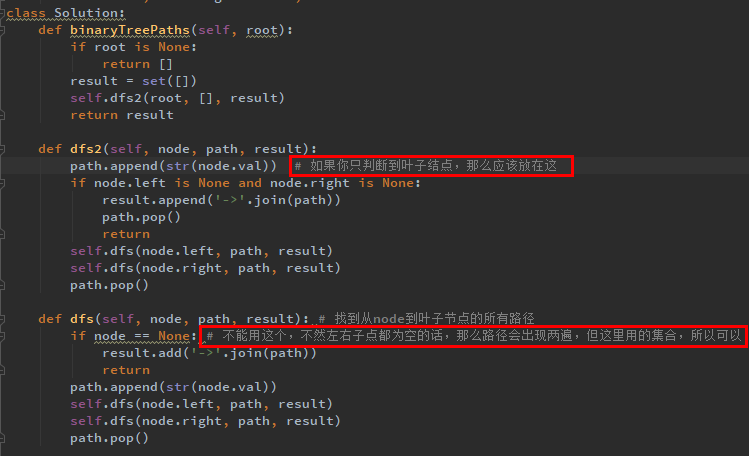
将二叉树后序遍历序列化



### 二叉树的所有路径

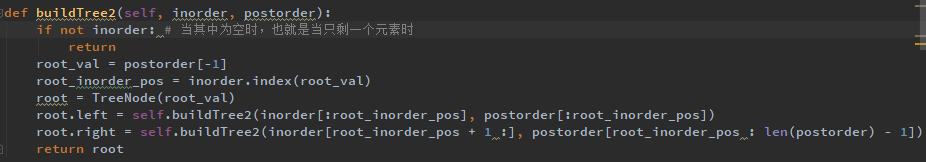
给出一棵二叉树，找出从根节点到叶子节点的所有路径

我的思路：传递一个字符串，进行遍历，到了叶子节点，记录该字符串，也就是记录一条路径。



### 中序遍历树和后续遍历树构造二叉树

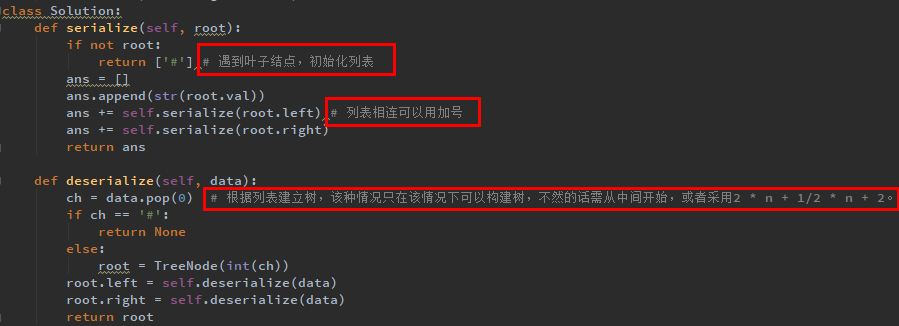
根据中序遍历树和后序遍历序列构造二叉树



### 二叉树的序列化和反序列化

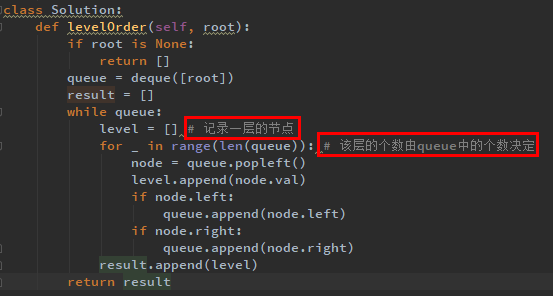
将数序列化成字符串，遇到空则返回‘#’，再根据该字符串反序列化为二叉树

我的思路：如果空节点用‘#’表示出来了，那么是可以根据序列化为字符串的遍历书序进行反序列化的。例如你用前序遍历序列化，那就可以用前序遍历反序列化。



### 二叉树的层次遍历1

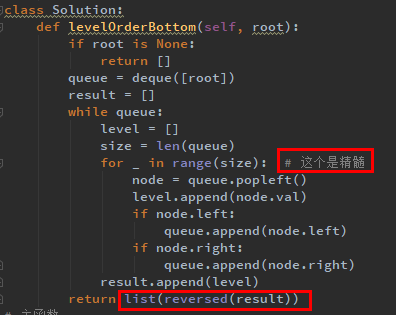
真正的层次遍历



### 二叉树的层次遍历2

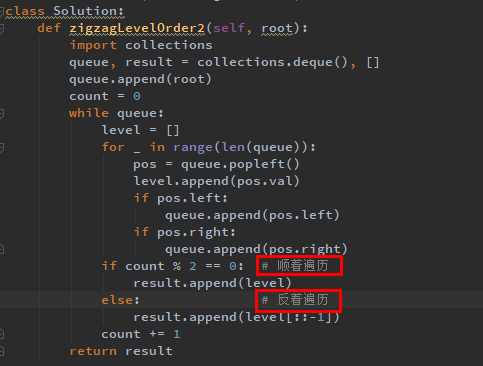
从下往上进行层次遍历

我的思路：应用上面的真正层次遍历思想

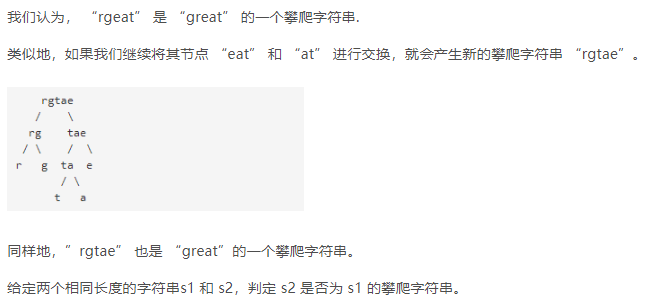
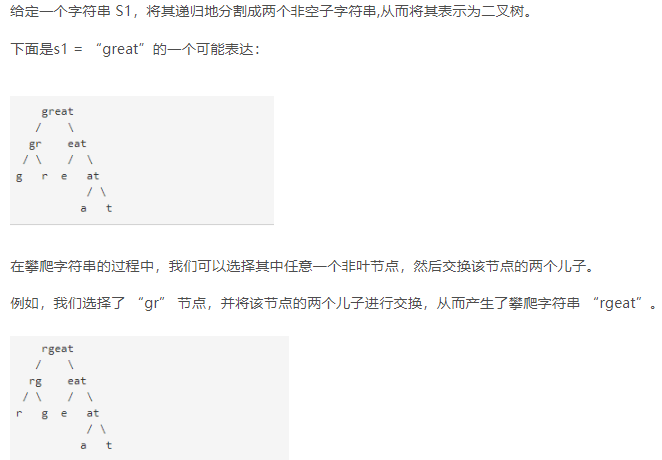


### 二叉树的锯齿形层次遍历

给出一棵二叉树，返回其节点值的锯齿形层次遍历，即先从左往右，下一层再从右往左，层与层之间交替进行。



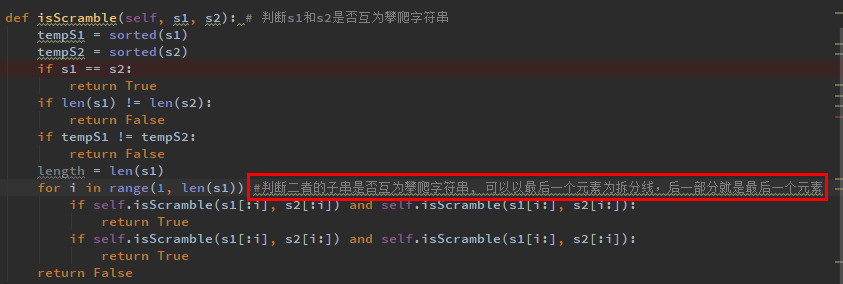
### 动态规划：攀爬字符串



我的思路：求出s1所有可能的攀爬字符串，看s2是否是。但这样太复杂。

解题思路：其实判断二者是否互为攀爬字串：

1. 二者是否长度相等，二者排序之后是否字母相同，否则，return False
2. 对截断位置进行遍历，假如将s1截断为s11,s12，将s2截断为s21,s22，如果s11 == s21 and s12 == s22 or s11 == s22 and s12 == s21，那么返回True
3. 那么出口是什么呢？出口应该是当s1 = s2时，返回return，也就是如果是互为攀爬字符串时，那么一定满足s11 == s21 and s12 == s22 or s11 == s22 and s12 == s21。

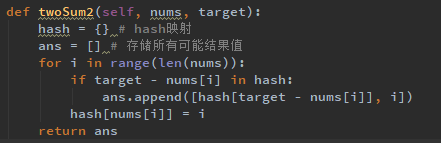


# 集合、数组、字符串

### 集合、列表、字符串：两数之和1

给出一个整数数组，找到两个数，使他们的和等于特定数target。返回两个数的下标，两个数不能重复。

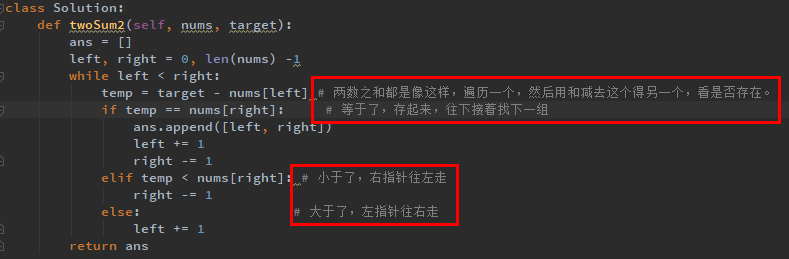
我的思路：定义一个集合hash，循环数组，target - A[i]是否在集合hash内，若在，则返回二者的下标[hash[target - A[i], i]。若不在，将nums[i]加入集合hash中。



### **集合、列表、字符串**：两数之和2

给定一个已经按升序排列的数组，找到两个数使它们的和等于特定数target，返回这两个数的下标。

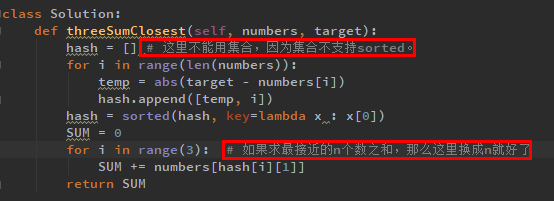
我的思路：相比于上面无序的数组，既然这是有序数组，那么就可以定义两个指针，从两边扫起，那这个时候就不需要映射集合hash了，因为有两个指针记录下标，如果只有一个，就需要利用hash映射进行另一个坐标的获取。这个就有点像二分法的变相运用。



### 集合、列表、字符串：最接近的三数之和

给出一个包含n个整数的数组s，找到与给定整数target最接近的三元组，返回这三元组的和。

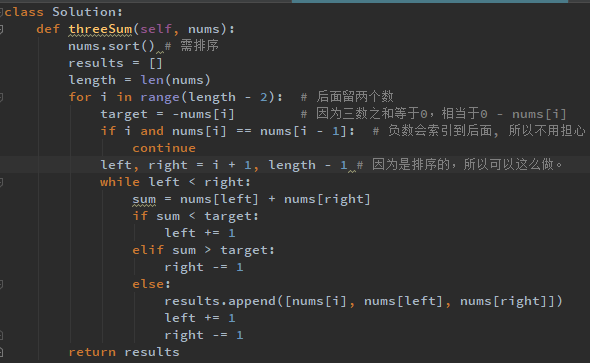
我的思路：将数组中的所有数都减去target，然后取绝对值，存入列表，一个绝对值，对应着一个下标，排序列表，求出最接近的三个数的下标，然后进行求和。



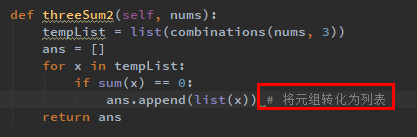
### 集合、列表、字符串：三数之和为0

给出一个有n个整数的数组S，在S中找到三个整数a、b、c，找到所有a + b + c = 0的元组。

我的思路：先遍历第一个整数，然后将剩余的两个数之和等于target - 第一个整数来做。



可以用combinations来做：

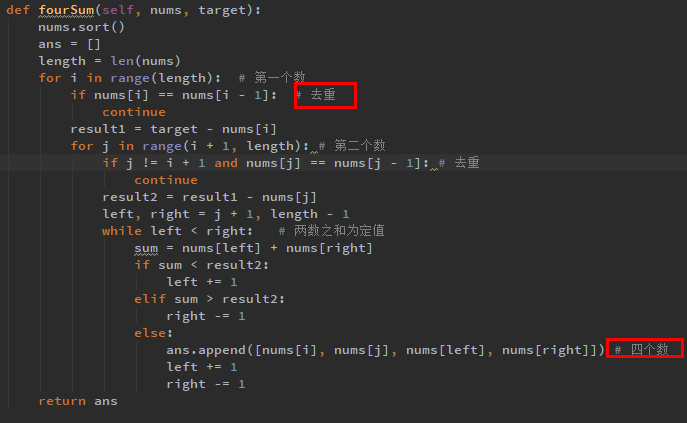


### 集合、列表、字符串：四数之和为定值

给一个包含n个数的整数数组，在s中找到所有使得和为给定整数target的四元组（a,b,c,d）

我的思路：直接用combinations就好

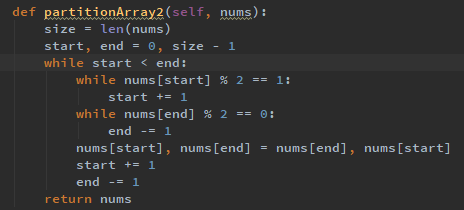
给出的代码过于复杂， 如下：



### 奇偶分割数组

分割一个整数数组，使得奇数在前，偶数在后。

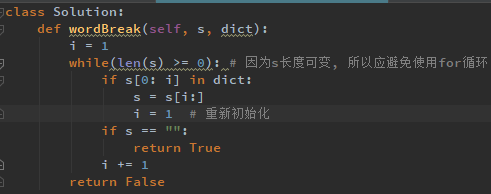
我的思路：快速排序的部分应用。左边找偶数，右边找奇数，交换。

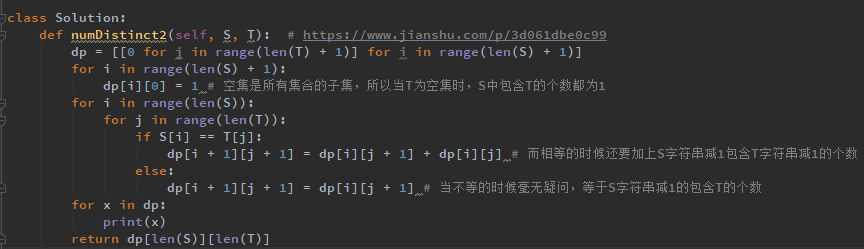


### 单词拆分1

给定字符串s和单词字典dict，本例将判断s是否可以分成一个或多个以空格分割的子字符串，并且这些子字符串都在字典中存在。

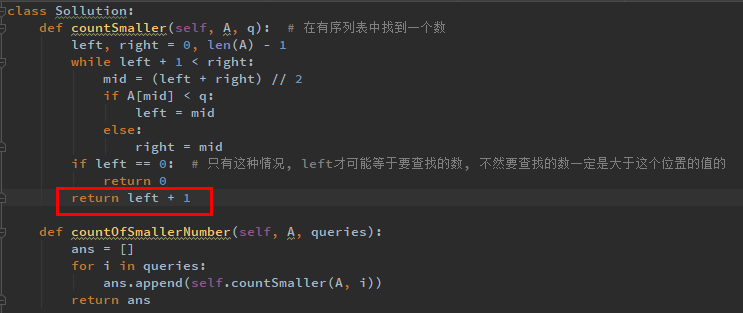
我的思路：截取0-i个字符作为新的字符串，如果字符串再字典中，那么就将后续未截取的字符串作为新的字符串进行重新的截取，知道字符串都空，返回True





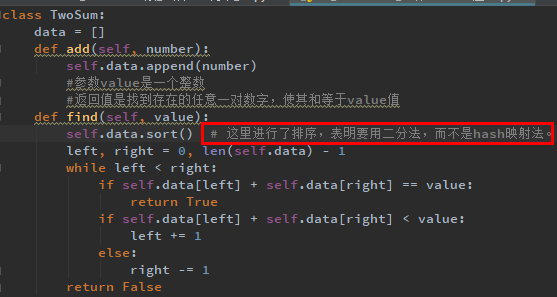
### 统计比给定整数小的数

给定一个整数数组（数组长度为n，元素的取值范围为0-10000），以及一个查询列表。每一个查询都会给出一个整数，本例将返回数组中小于该给定整数的元素数量。



### 两数之和3

设计并实现一个TwoSum类，需要支持add和find操作，add操作把这个数添加到内部的数据结构，find操作判断是否存在任意一对数字之和等于这个值。



# 矩阵图形

### 矩阵图形：太平洋和大西洋的水流

给定一个m行n列的非负矩阵代表一个大洲，矩阵中每个单元格的值代表此处的地形高度，矩阵的左边缘和上边缘是太平洋用‘~’表示，下边缘和右边缘是大西洋用‘\*’表示。

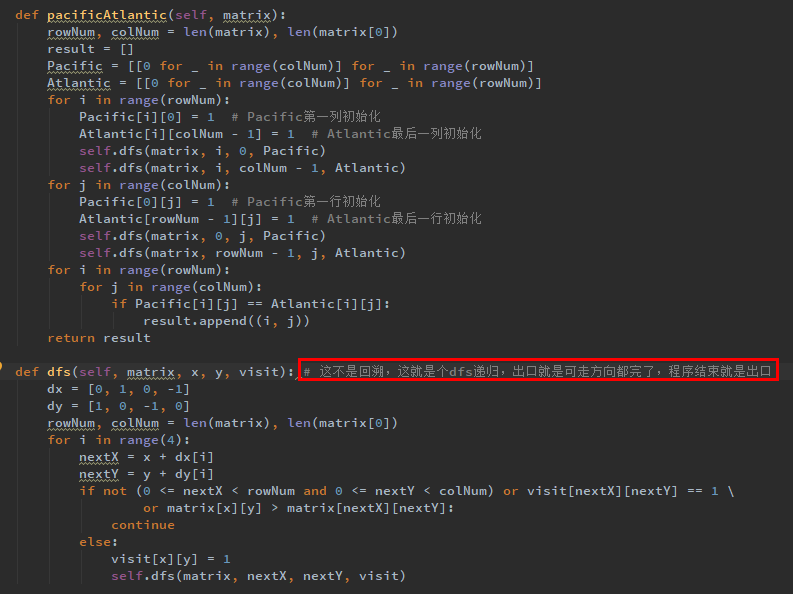
水流只能在四个方向（上、下、左、右）从一个单元格流向另一个海拔与自己相等或比自己低的单元格。本例将找到那些从此处出发的水既可以留到太平洋，又可以留到大西洋单元格的坐标。

我的思路：因为从中间取值明显不好取，所以肯定是从四周往中间反向比较好。也就是

设立两个数组Pacific和Atlantic，分别表示从太平洋和大西洋出发，经过的 地理位置，并沿途置1，最终遍历两数组，两数组的某一位置两个数组都为 1的情况那么就是既能流到太平洋又能流到大西洋的情况。

具体算法实现为：

1. 创立两个和原列表一样大的列表
2. 建立一个函数，专门遍历四周起始点，并以pacific[x][y]和Atlantic[x][y]为参数传入dfs函数
3. 建立dfs函数，所需的参数有：起始点、访问数组，将起始点置1，表示经过，如果遇到出界或者matirx[x][y] > matirx[nextX][nextY]的情况或者遇到了pacific[nextX][nextY]/Atlantic[nextX][nextY] == 1的情况则换个方向。



### 矩阵图形：不同岛屿的个数

给定一个由0和1组成的非空二维网格代表一个岛屿，一个岛屿是指四个方向（包括恒向和纵向）都相连的一组1， 其中1表示陆地，0表示水域。假设网格的四个边缘都被水包围了，找出所有不同岛屿的个数，如果一个岛屿可以被转换（不考虑旋转和翻折）成另一个岛屿，则认为两个岛屿是相同的。

我的思路：1、找到不同岛屿的左上角的点，找到岛后，把所有的岛的1置0。

2、将岛屿中的点都一边放入queue，一边弹出最左边点，与岛屿的左上角 点进行横纵坐标相减，然后将差转化为字符连接起来

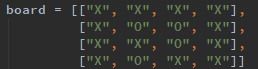
3、字符串相等的岛就是相同的岛。



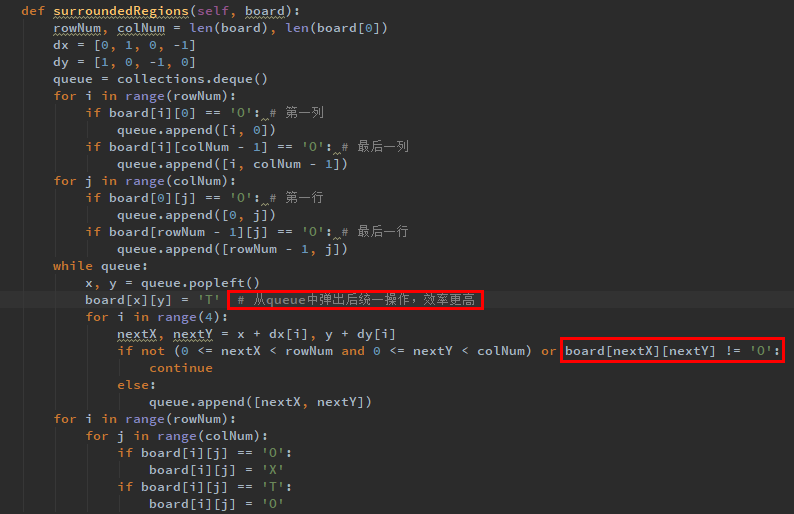
总之，queue可以解决局部遍历的问题，而回溯只能全局遍历。像这里可以只遍历岛屿，不遍历水域。

### 矩阵图形：被围绕的区域

给定一个二维矩阵，其中包含‘X’和‘O’，本例将找到所有被‘X’围绕的区域，并用‘X’填满。



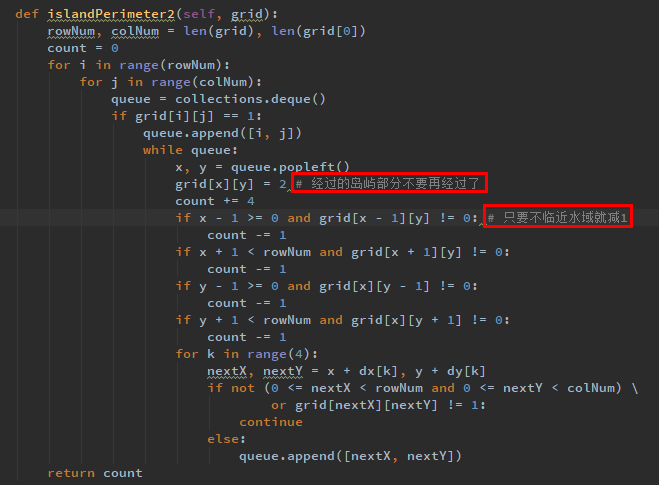
我的思路：题目的意思是填满所有被‘X’围绕的‘O’区域，这里的思路就是你要从中间把O替换为X，但是你与不知道该O区域能否连通外界，所以你没法直接替换。所以你想思维，就是你从四周找O，然后将所有O区域找出来，并将O区域更改为T区域，最后在整个遍历一遍列表，将所有的O替换为X，将所有的T更改回O。



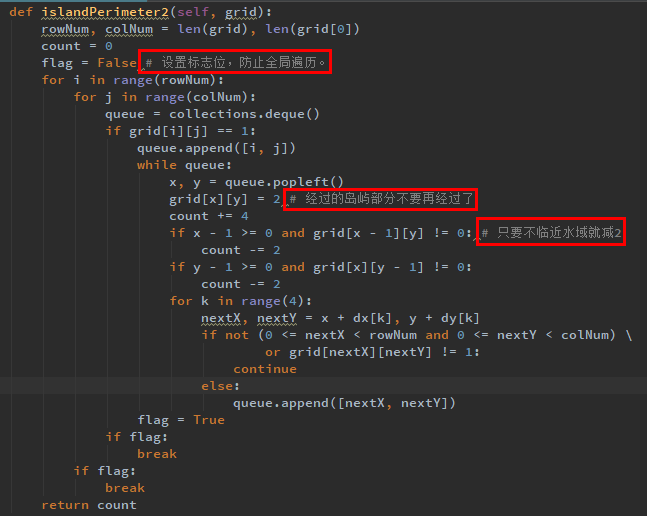
### 思路新颖：矩阵图形：岛的周长问题

给定一张用二维数组表示的网格地图，其中1表示陆地，0表示水域单元格，网格地图中的单元格都为水平、垂直相连（斜向不相连）。这个网格地图四周完全被水域包围，而且其中有且仅有一个岛（岛定义为一块或多块相连的陆地单元格），这个岛不包含湖（湖定义为不和外围水域相连的水域单元格）。一个地图单元格是一个边长为1的正方形；网格地图是一个矩形，并且它的长和宽不超过100，本例将求出这个岛的周长。

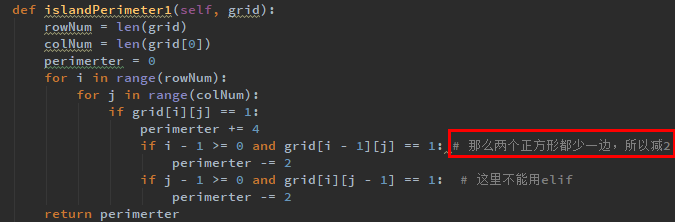
我的思路：找到这个岛最左上方的那个点，然后加入queue，利用queue遍历这个小岛，思路是，首先遇到1就加4，然后看这座小岛几面与岛的内部相邻，那么就减几。



改进如下：



上面使用了queue进行局部遍历，下面直接用全局遍历：

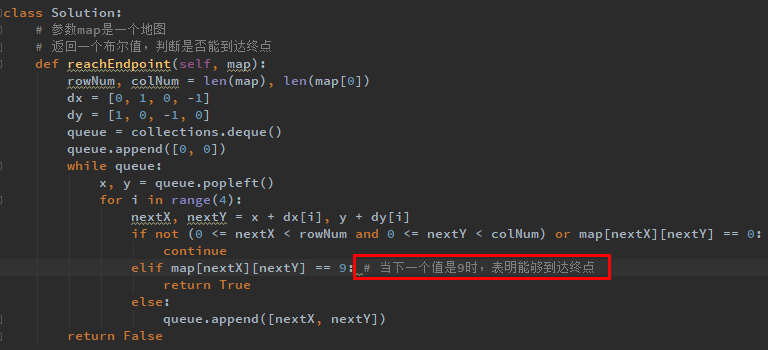


### 矩阵图形：能否到达终点

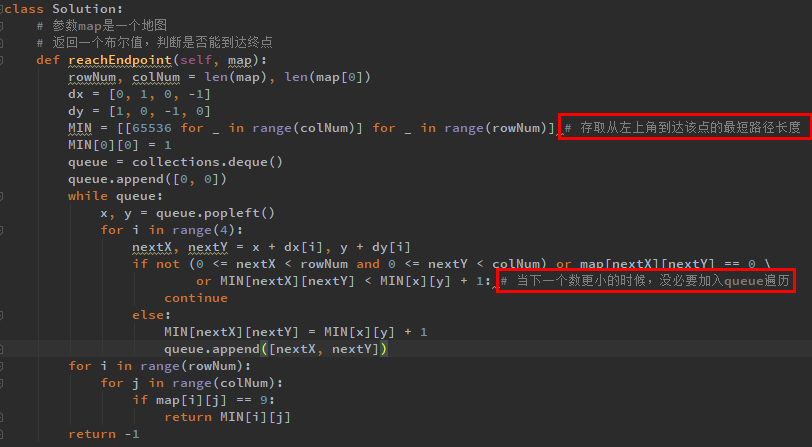
给定一个m行n列的矩阵表示地图，1代表空地，0代表障碍物，9代表终点，判断从（0,0）开始能否到达终点，若能到达终点放回True，否则返回False

我的思路：之前有过一道类似的题目就是219迷宫1，但是其是用回溯的方法解决的，这里尝试着用双端队列的方式解决该问题。

从左上方走起，将左上方点的位置放入queue中，当下一个要放入queue中的位置的元素是9时，那么就返回True。



如果要用deque求最短路径的话。

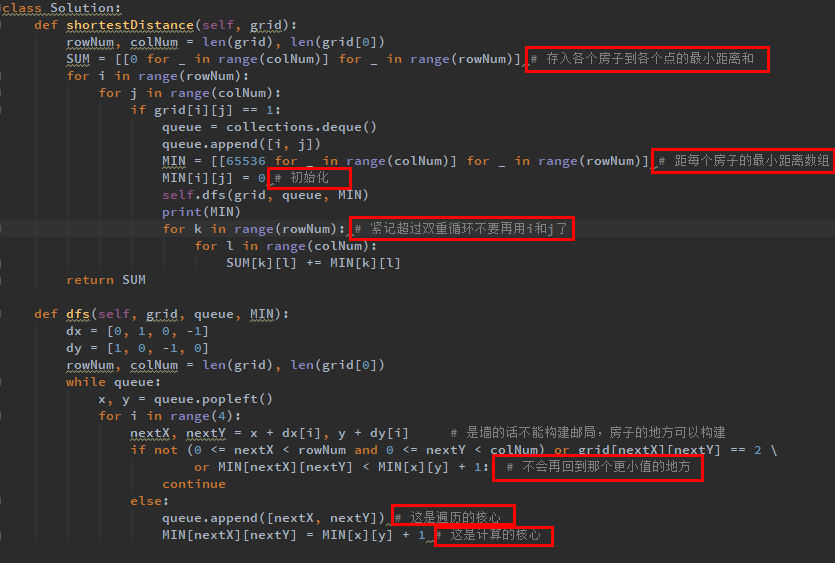


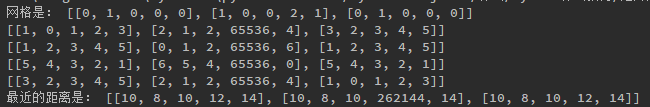
### 矩阵图形：建立邮局

给出一个二维网格，每一格可以代表墙（2）、房子（1）和空（0），在网格中找到一个空的位置建立邮局，使得所有房子到邮局的距离和是最小的。返回所有房子到邮局的最小距离和，如果没有地方建立邮局，则返回-1.

解题思路：一般这种问题用queue解决会容易很多，因为可以用queue计算出每个点离出口的最短距离，所以这里找出网格中为1的点，然后求出各个位置到其的最短距离，用一个统一的sum数组，将各个位置的最短距离存入sum数组，到时找到sum数组中最小值的位置，就是建立邮局的位置。

可以用MIN数组代替visit数组。因为遇到更小的数不会回去了。而且又计算了最小距离，一举两得。





所有最近距离和是8，其中两个是建立在房子旁边的。

### 矩阵图形：迷宫1

一个二维数组表示一个迷宫，0表示为空，1表示为路障，给定任一个位置，人是否可以走出迷宫，迷宫出口默认在二维数组右下角：

我的思路：回溯四个方向，默认为右，下，左，上，并设置visit数组，给走过的位置置1，设下路障。对于下一个坐标（nextX， nextY）

If 越界 或者 遇到路障：

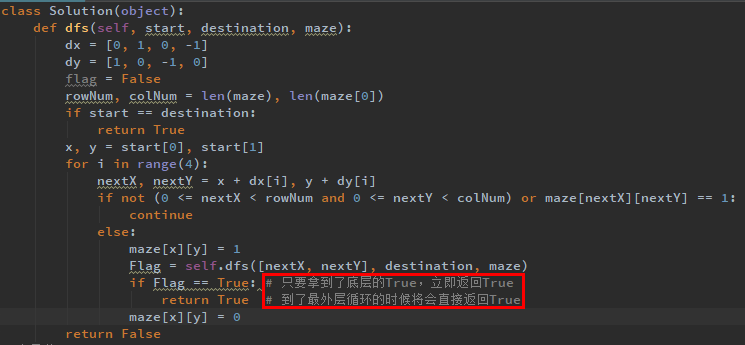
Continue(换一个方向)

else：

x, y = nextX, nextY

进入下一层递归。

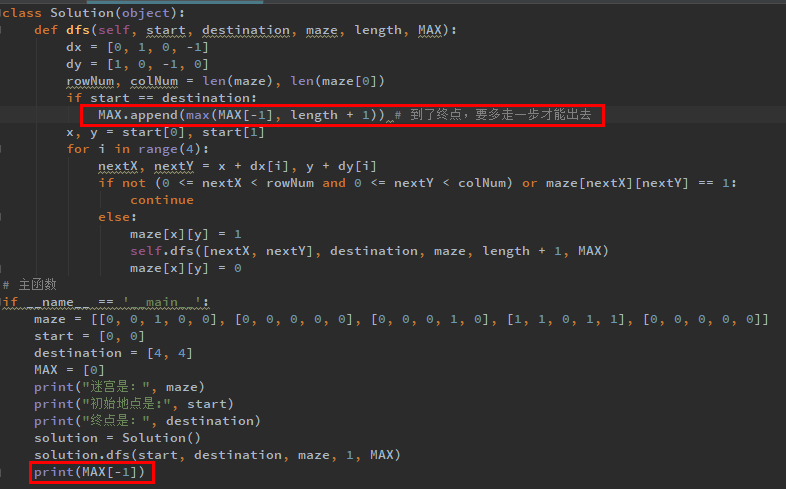
所以由上推断，递归的出口时x，y等于出口坐标。



### 矩阵图形：迷宫2

相比于1，要找出走出迷宫的最短距离。

我的思路：传递一个值，计算路径的长度，传递一个数组，路径的最短值。



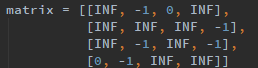
### 矩阵图形：迷宫3

给定三个值初始化的m行n列的二维网格，-1表示墙壁或障碍物，0表示门，INF表示空房间，使用值2的31次方 - 1 = 2147483647表示INF，假设到门的距离小于2147483647。在代表每个空房间的网络中填入与门最近的距离，如果不可能到达门口，则应填入INF。

我的思路：如果用queue来操作的话，从左上角开始操作，那么就错咯。

解题思路: 将门的位置都放入queue，然后先弹出第一扇门，将这扇门周围为空位置的地方赋值为门所在位置数加1，表明距离，把空位置当做另一扇门，再次放入queue中，再寻找周围为空位置的地方，循环往复。空位置的地方的值始终要保持最小哦。

总之一句话就是利用queue进行遍历，相比于利用回溯遍历，前者可以有多个入口，并且可以比较最值。

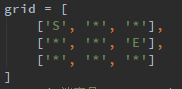


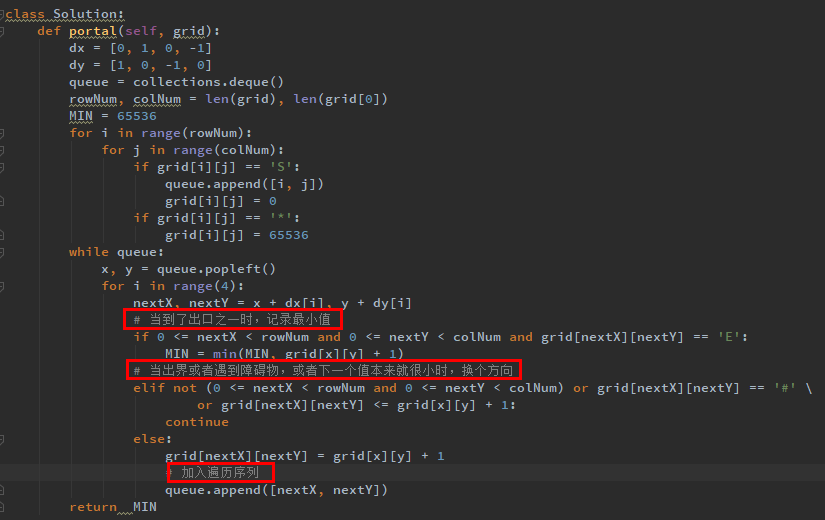


### 矩阵图形：迷宫4

给定一个m行n列的二维字符数组表示迷宫。它有四种房间，‘S’代表从哪开始（只有一个起点），‘E’代表迷宫的出口（当抵达出口时，将离开迷宫，该题目会有多个出口），

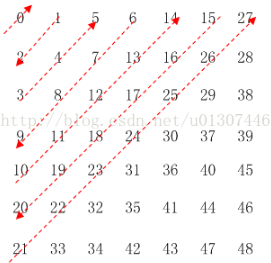
‘\*’代表这个房间可以经过，‘#’代表一堵墙，不能经过墙。每次可以上下左右移动到达一个房间，花费一分钟时间，但是不能到达墙。本例将得出离开这个迷宫所需的最少时间，如果不能离开，则返回-1.





### 矩阵图形：矩阵元素ZigZag返回

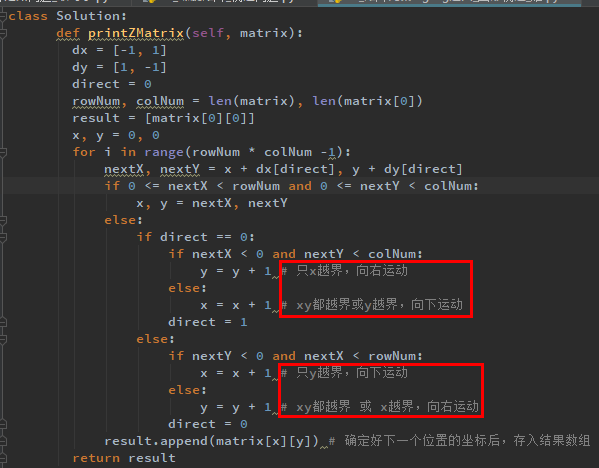
给定一个m行、n列的矩阵，以ZigZag的顺序返回矩阵中所有元素，如图所示：



我的思路：其实这道题感觉和54题螺旋矩阵差不多，只不过是两个方向（向右上，和向左下），越界了，就向右或向下一步，然后换个方向继续，这里不会走回头路。

如果方向是右上，只x越界，向右运动，否则向下运动；

如果方向是左下，只y越界，向下运动，否做向右运动。



### 矩阵图形：螺旋矩阵

给定一个m行n列的矩阵，本例将按照螺旋顺序返回该矩阵中的所有元素。螺旋顺序从图1中左上角元素开始，遍历所有元素的顺序。

我的思路：这道题和回溯没有关系，就是一道纯考思路的题，首先要确定四个方向，如果没有越界或者没有走回头路，那么就一直沿着这个方向下去，并沿途对visit[x][y]置1，对于下一个坐标（nextX，nextY）:

If 没有越界 且visit[nextX][nextY] == False：

保持该方向移动，下一个位置就是（nextX，nextY）

else

如果方向向右，x + 1向下移动，下一个位置就是（x + 1， y），并将方向改为向下。

如果方向向下，y - 1向左移动，下一个位置就是（x， y - 1）并将方向改为向左

如果方向向左，x - 1向上运动，下一个位置就是（x - 1， y）并把方向改为向上

如果方向向上，y + 1向右运动，下一个位置就是（x， y + 1）并把方向改为向右

