Debug点：

词典是不是因为重复值被覆盖

是不是有重复值

是不是有孤立点

输入是一组还是多组

有没有考虑0，不止一位数，负数，空

注意节点序号是从0开始还是从1开始

用到优先队列的时候，一定要把weight放在第一位

行列有没有弄反

索引是不是从0开始！！！！尤其在树，图的问题中，一定注意

语法点：

浅拷贝和深拷贝，浅拷贝是全部相互影响，深拷贝互不影响

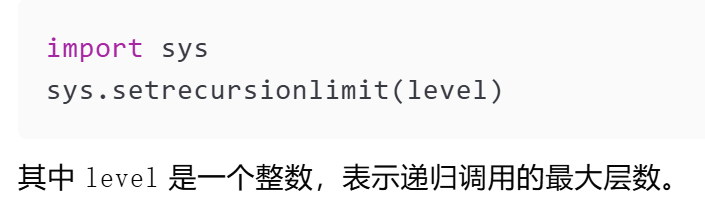
import copy

ss=copy.copy(ii)

dd=copy.deepcopy(ii)

.第一行加 # pylint: skip-fle 可以忽略检查

递归深度调整：





if char.isalpha(): # 如果是字母

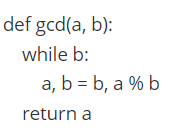
‘123’isdigit() #是不是数字

islower(),isupper()

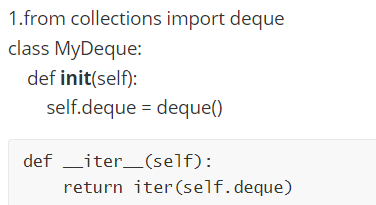
自定义标准排序：

candies=sorted(candies,key=lambda x:x[0]/x[1],reverse=True)

辗转相除算最大公约数：



自定义可迭代的deque：



不定参数接受：

op, \*args = map(int, input().split())  
args此时是一个列表

iterable.count(value)

str.find(sub) #未找到抛出-1

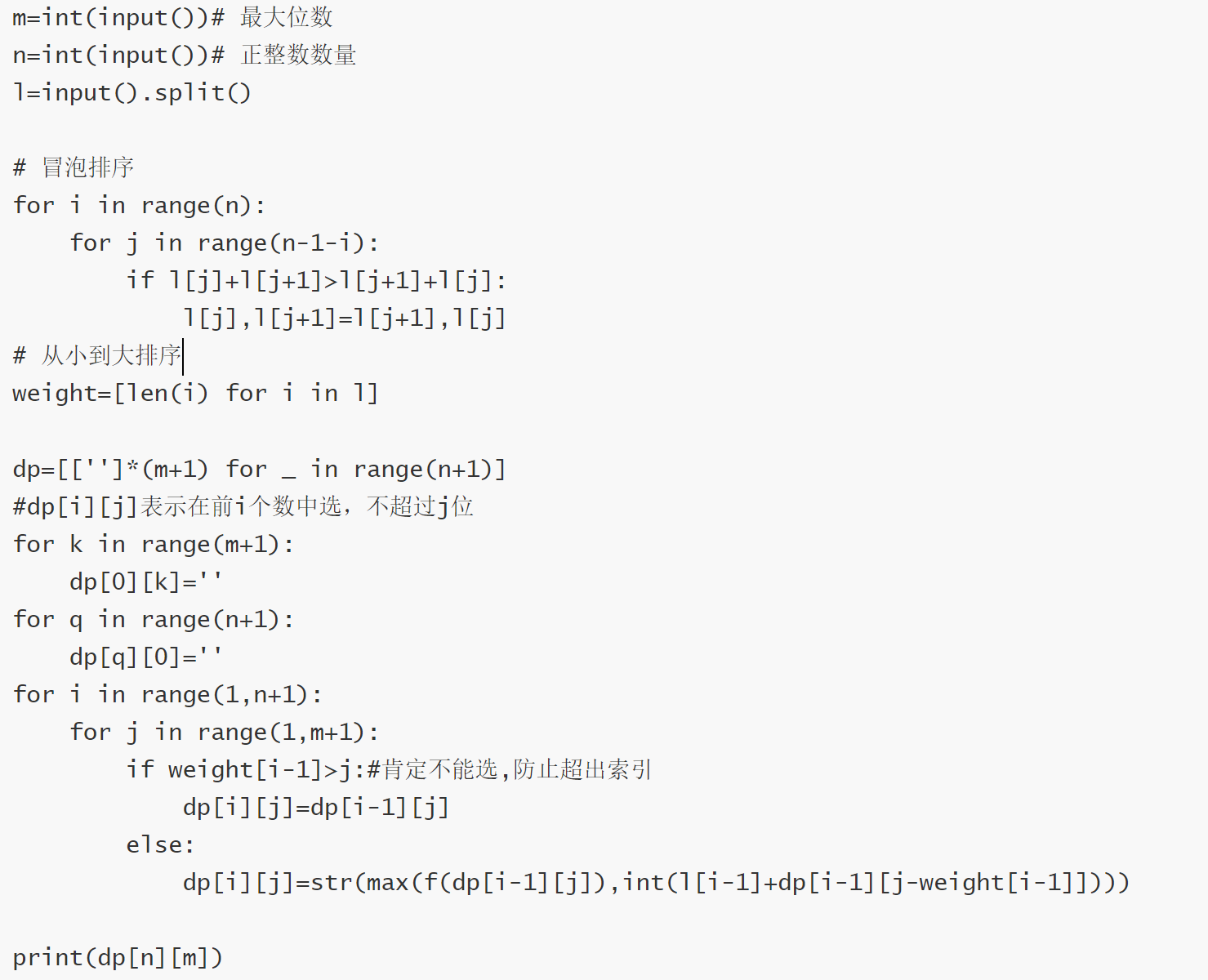
list.index(x) #未找到抛出ValueError

replace() 替换字符串中的指定字符， eval() 函数计算表达式的值

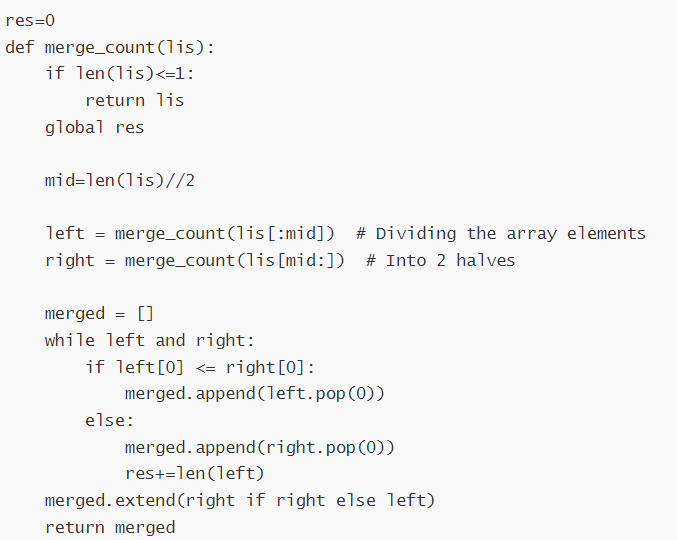
enumerate 快速获取索引和值： for index, value in enumerate(list, start)

冒泡排序应用：

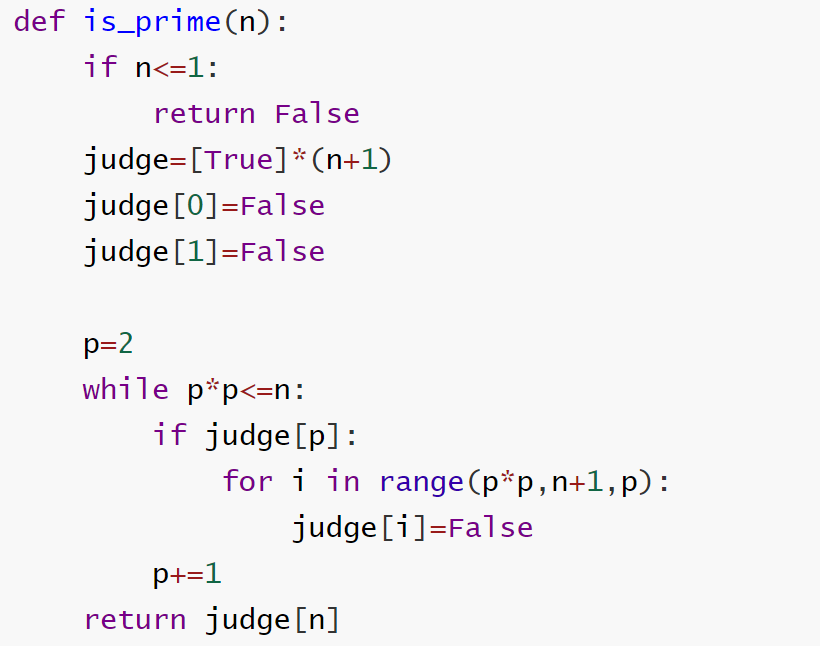
n个数组合，求最大的数，不超过多少位



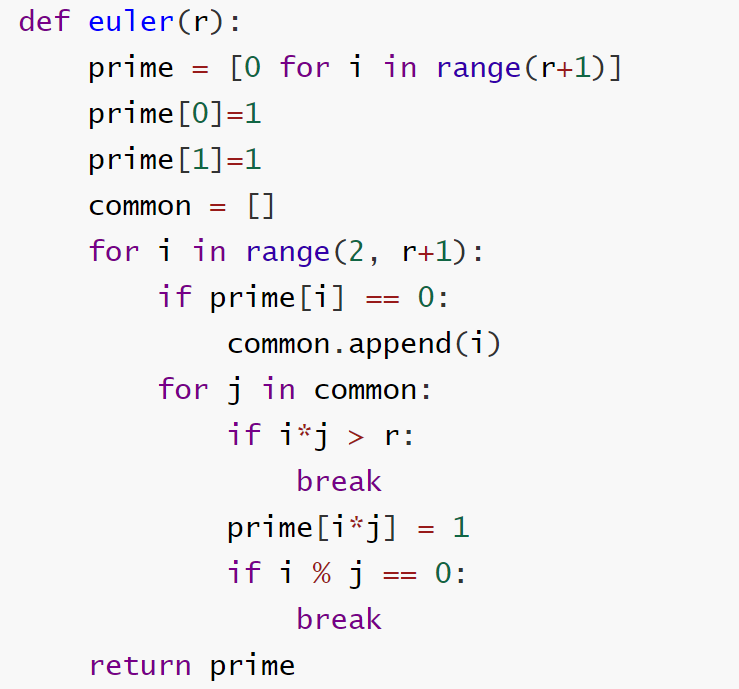
归并排序：



埃氏筛法：



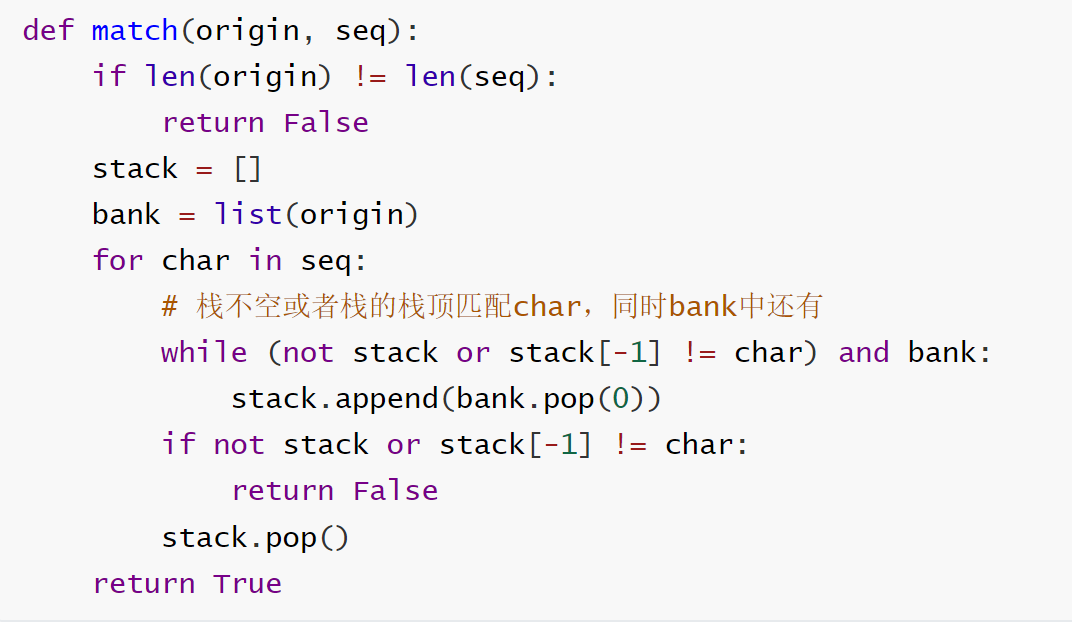
欧式筛法：



知道树的前序和后序遍历，求有几种可能：



合法出栈序列：



中置表达式转后置表达式（如何处理小数）



动态中位数：

两个堆，一个模拟左子树，一个模拟右子树：

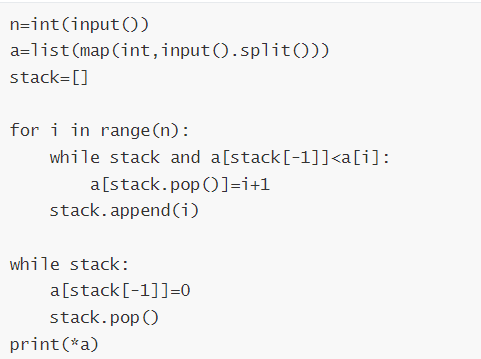


八皇后：

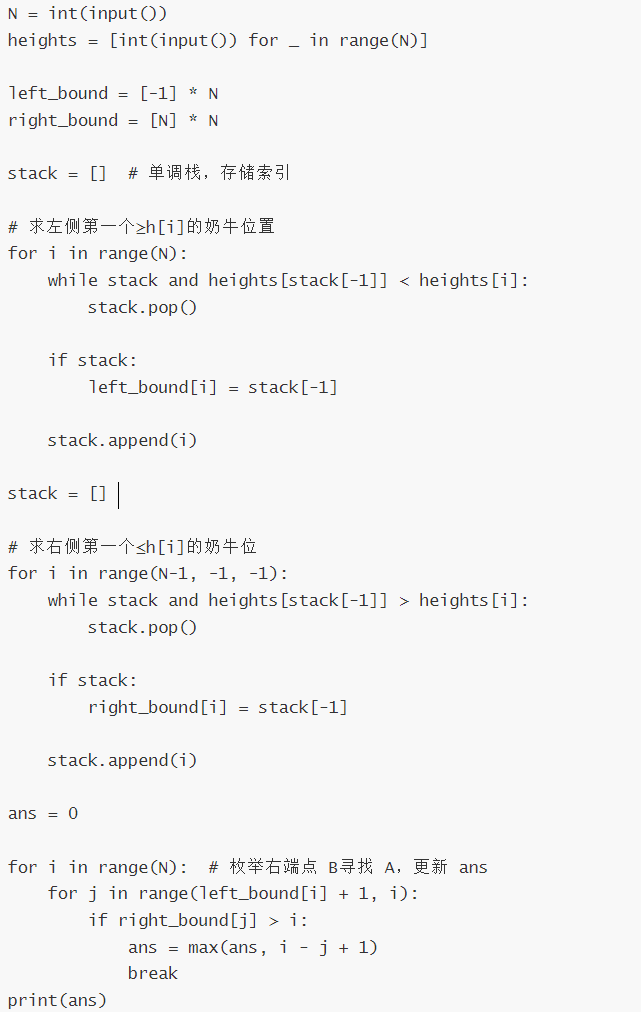


单调栈：

给出项数为 n 的整数数列 a1...an。定义函数 f(i) 代表数列中第 i 个元素之后第一个大于 ai 的元素的\*\*下标\*\*，。若不存在，则 f(i)=0。试求出 f(1...n)

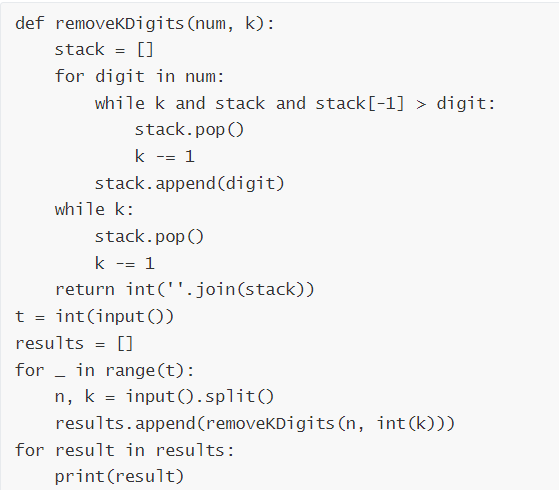


奶牛排队：

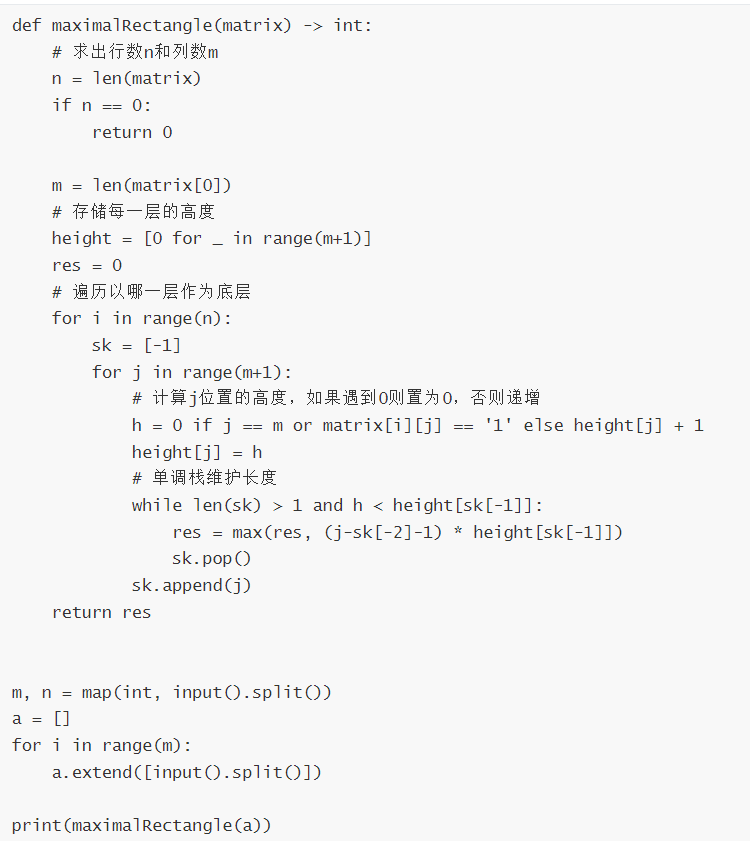


最小新整数：

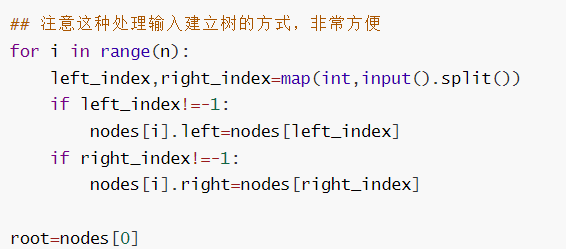
给定一个十进制正整数n(0 < n < 1000000000)，每个数位上数字均不为0。n的位数为m。 现在从m位中删除k位(0<k < m)，求生成的新整数最小为多少？ 例如: n = 9128456, k = 2, 则生成的新整数最小为12456



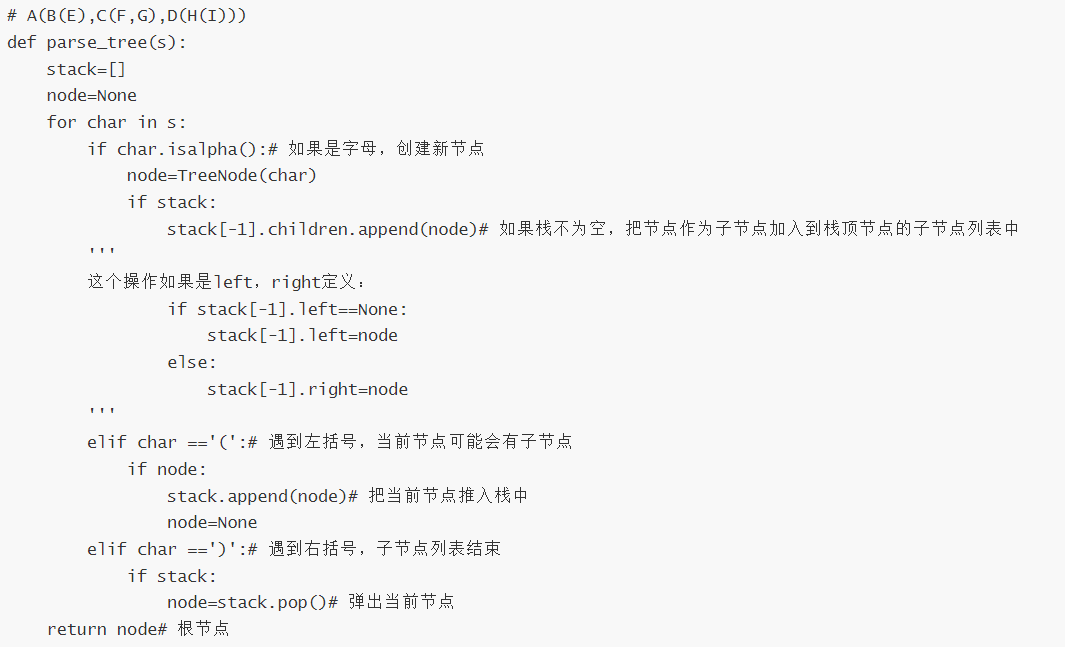
护林员盖房子：在一片保护林中，护林员想要盖一座房子来居住，但他不能砍伐任何树木。 现在请你帮他计算：保护林中所能用来盖房子的矩形空地的最大面积。子矩阵边长可以为1，也就是说： 0 0 0 0 0 依然是一个可以盖房子的子矩阵。



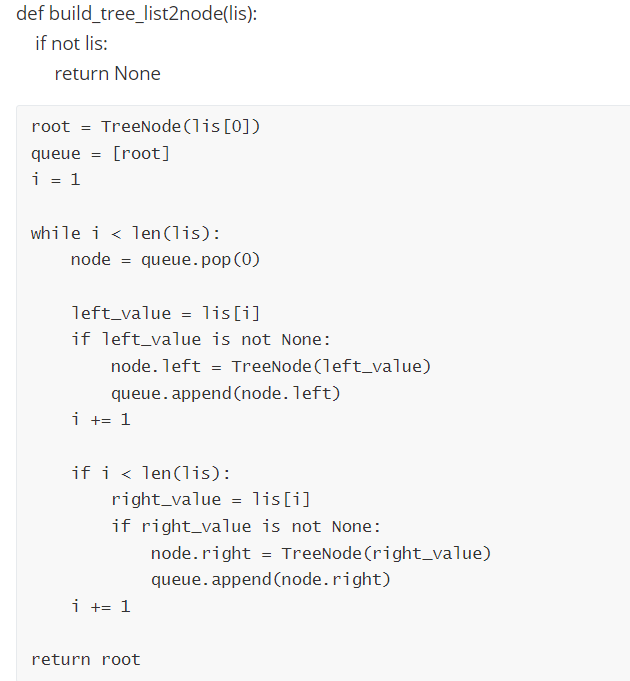
树



括号嵌套表达式生成树：

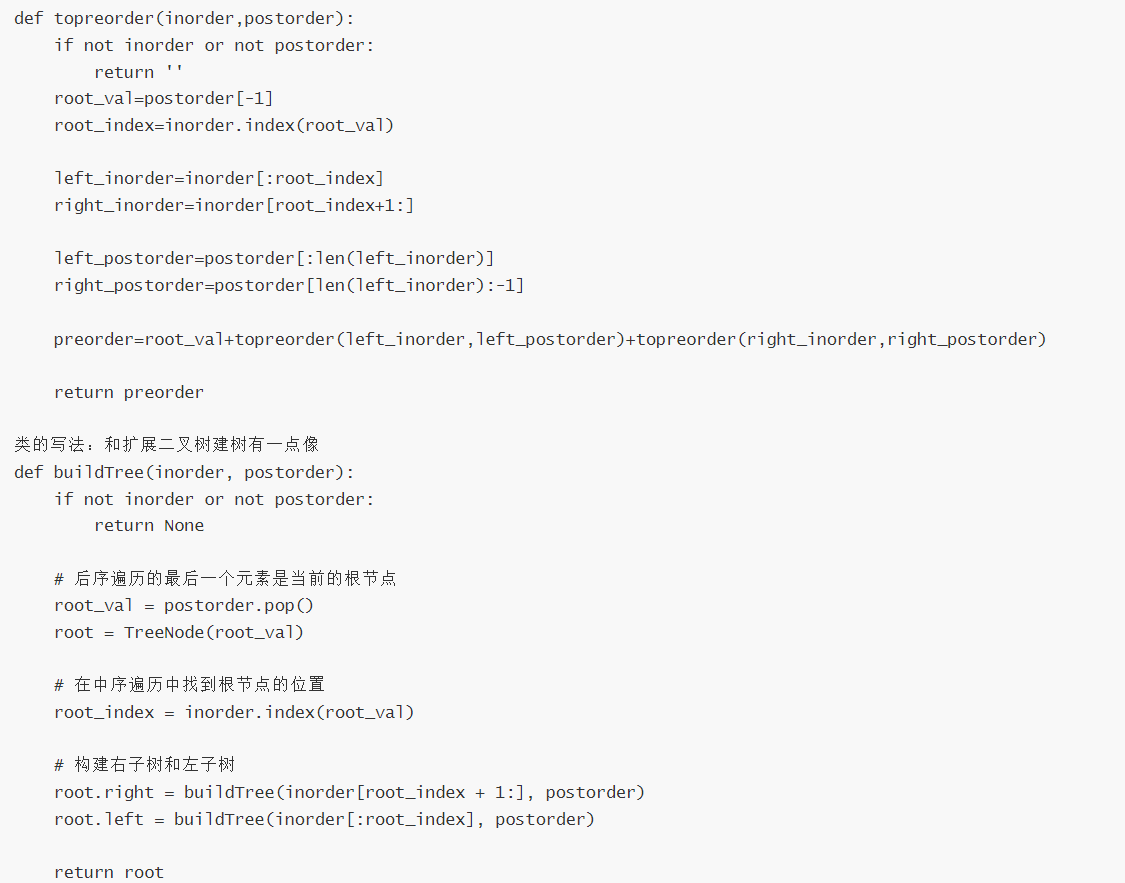


层次遍历树生成node树：



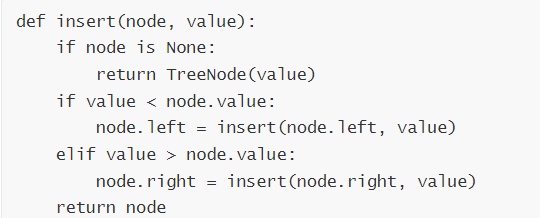
一般扩展二叉树，都可以考虑用递归

二叉树中序后序，建树：



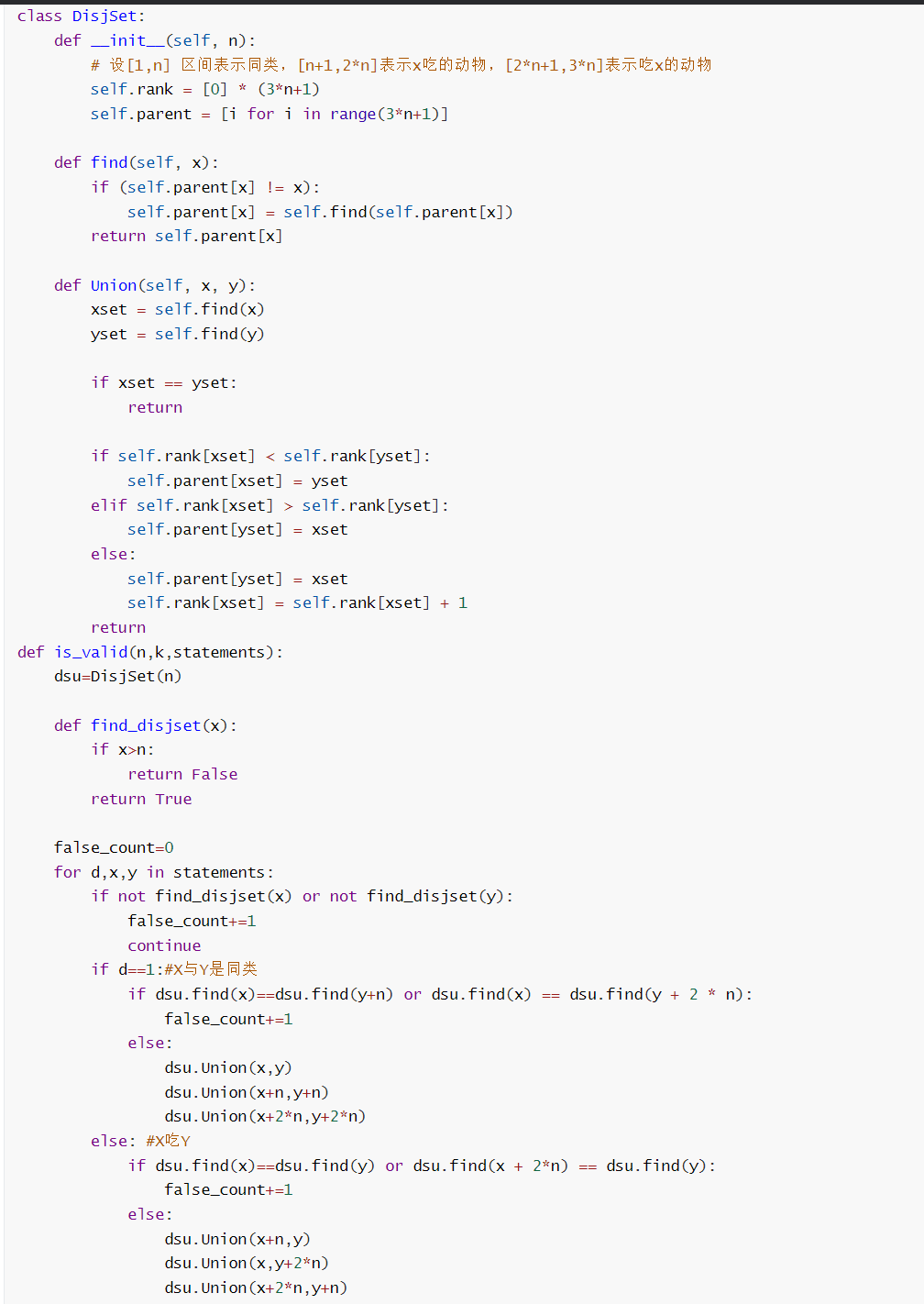
二叉搜索树的中序遍历，就是sort排序

给一行数，建立二叉搜索树：

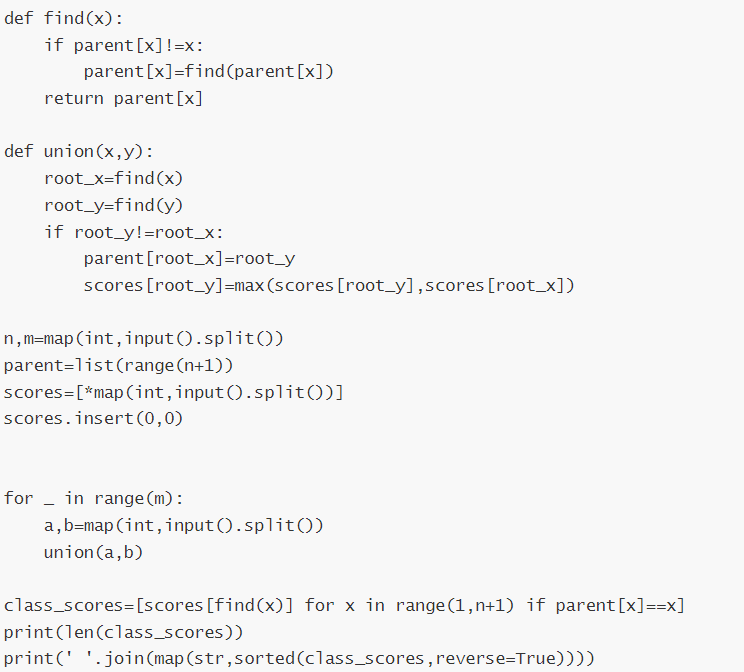


并查集：

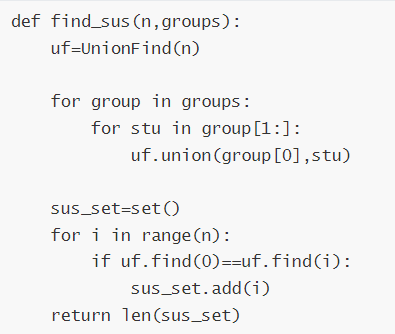
食物链：



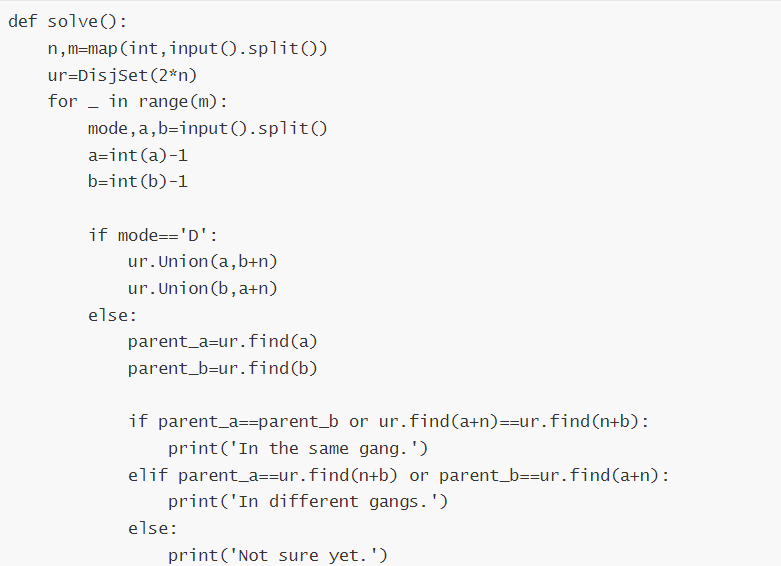
班级最高分：



找suspect：



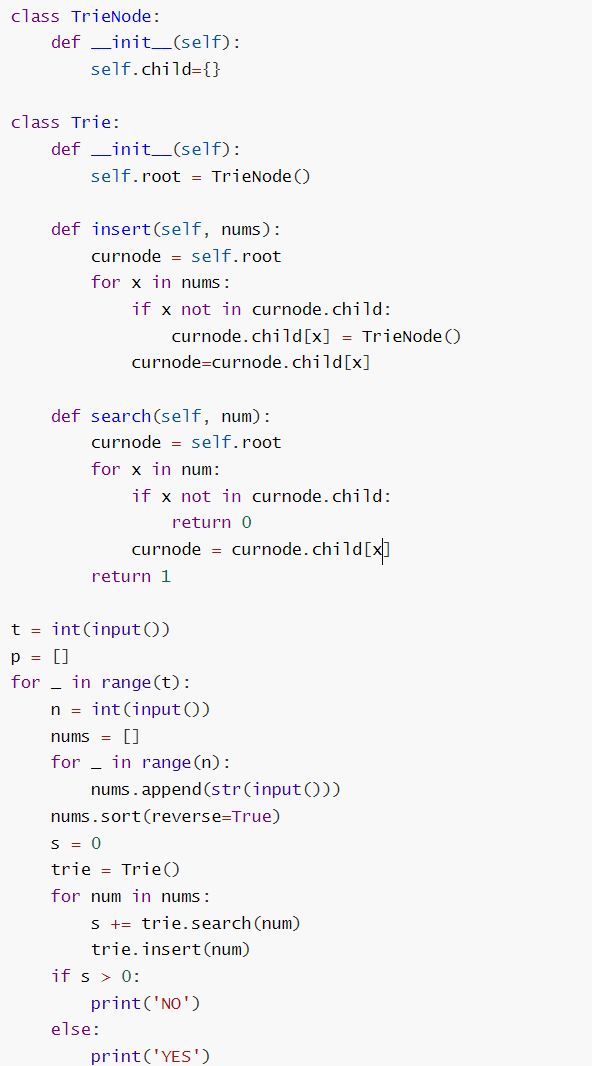
发现他，抓住他：



遍历树，输出是按照当前节点和子节点从小到大的顺序：



字典树：电话号码查找：

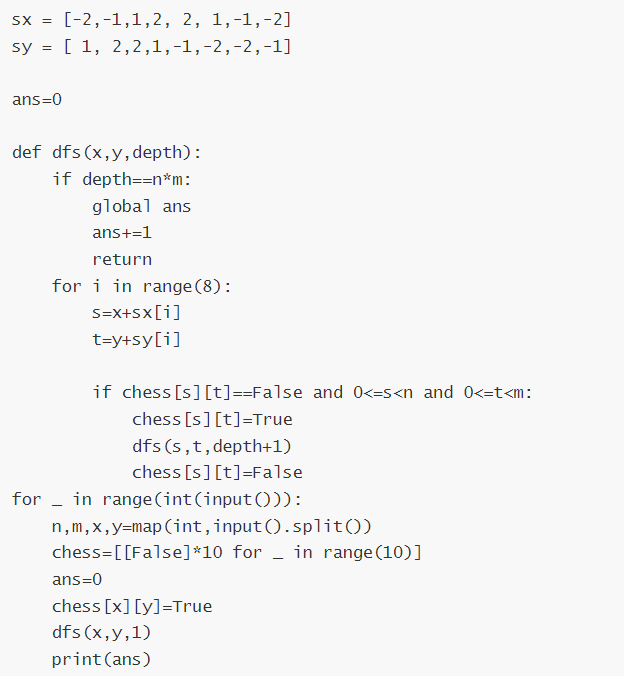


图

词梯：BFS



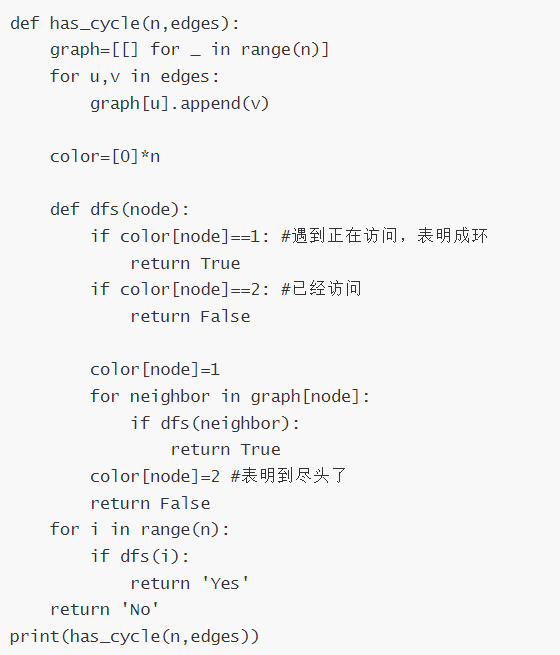
马走日：DFS



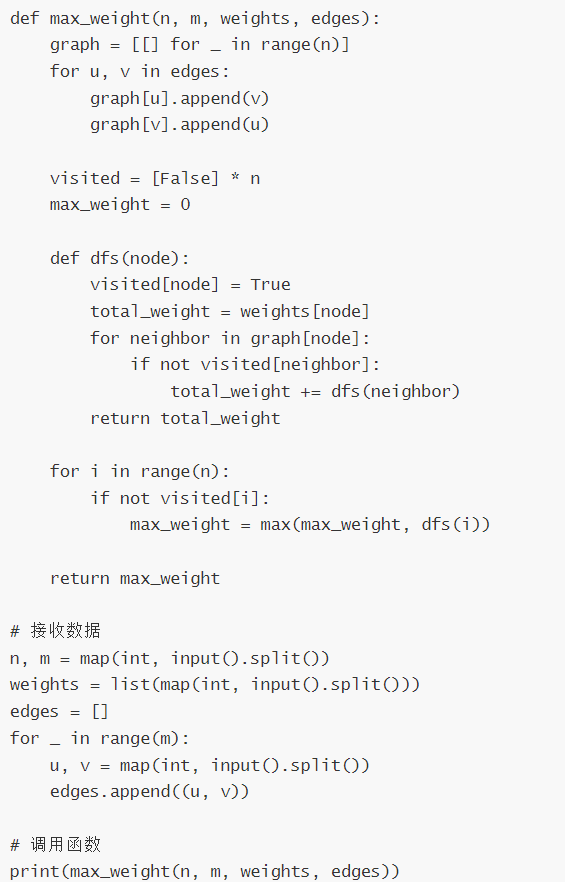
无向图联通快个数：



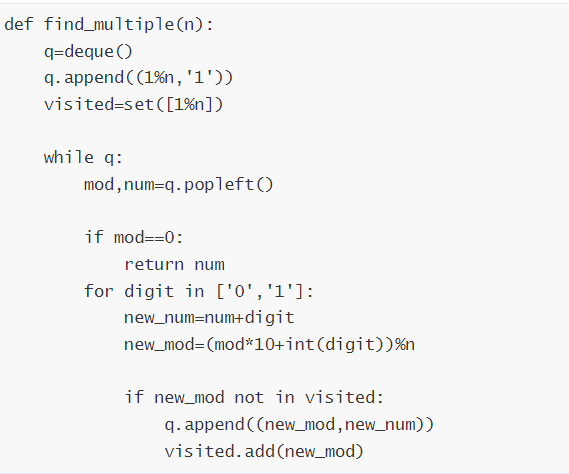
有向图判环：



无向图最大权值联通块：

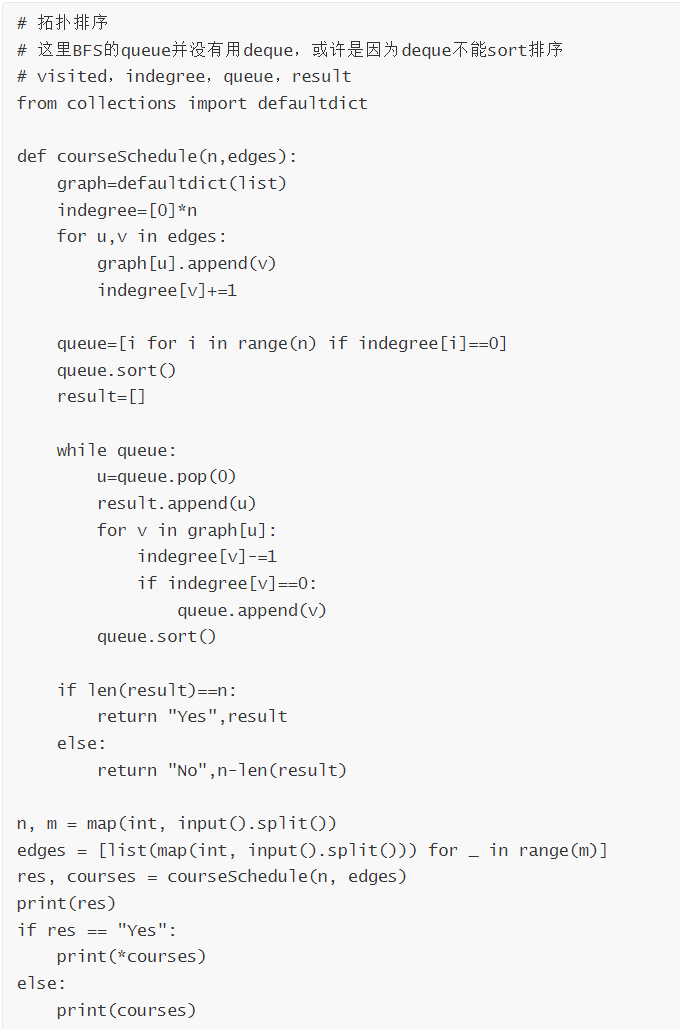


找0-1组成的倍数：



拓扑排序：

先导课程：



Sorting it all out：想清楚是中途判断的任务还是结尾判断



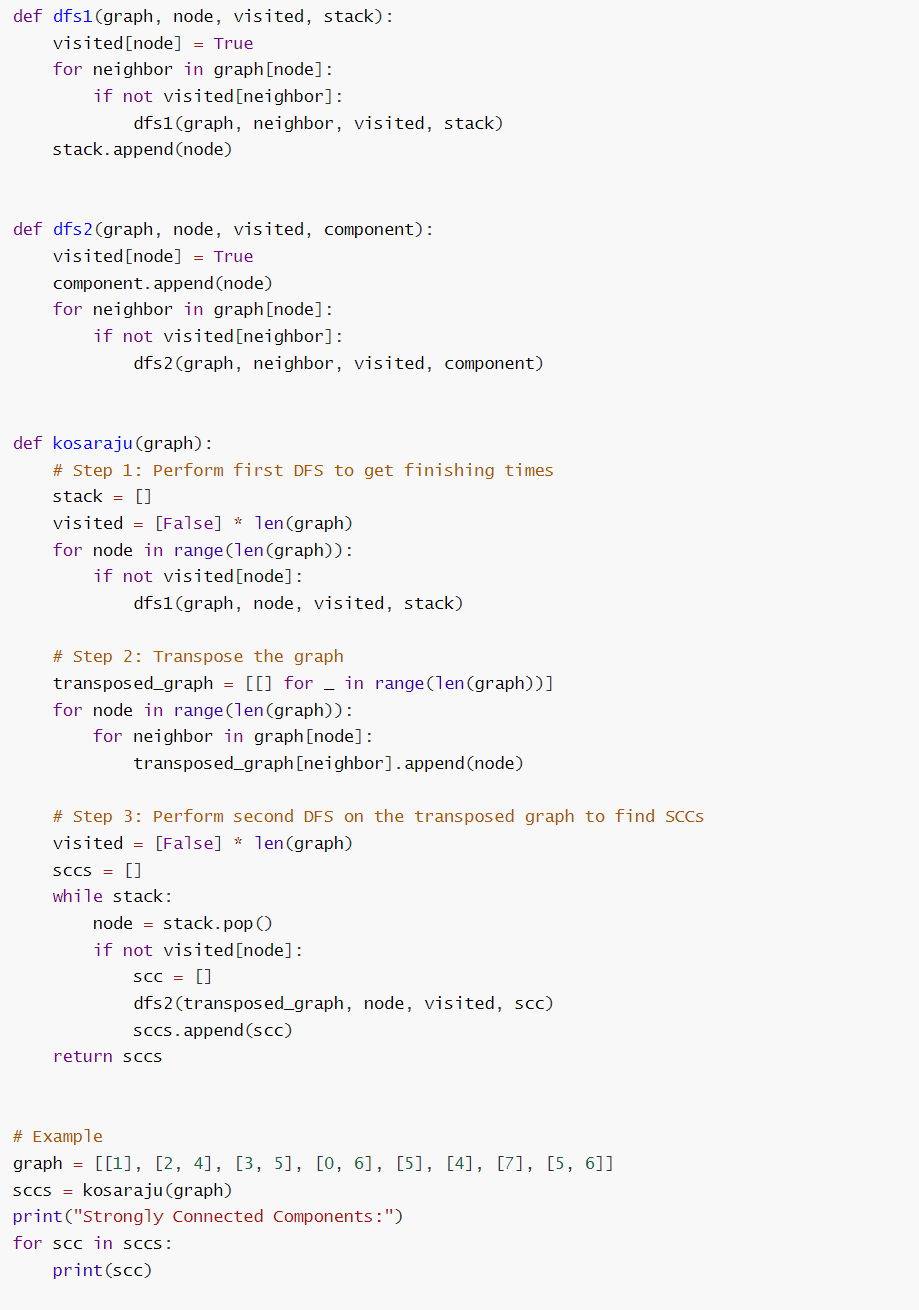
强连通单元：

Kosaraju算法的核心思想就是两次深度优先搜索（DFS）

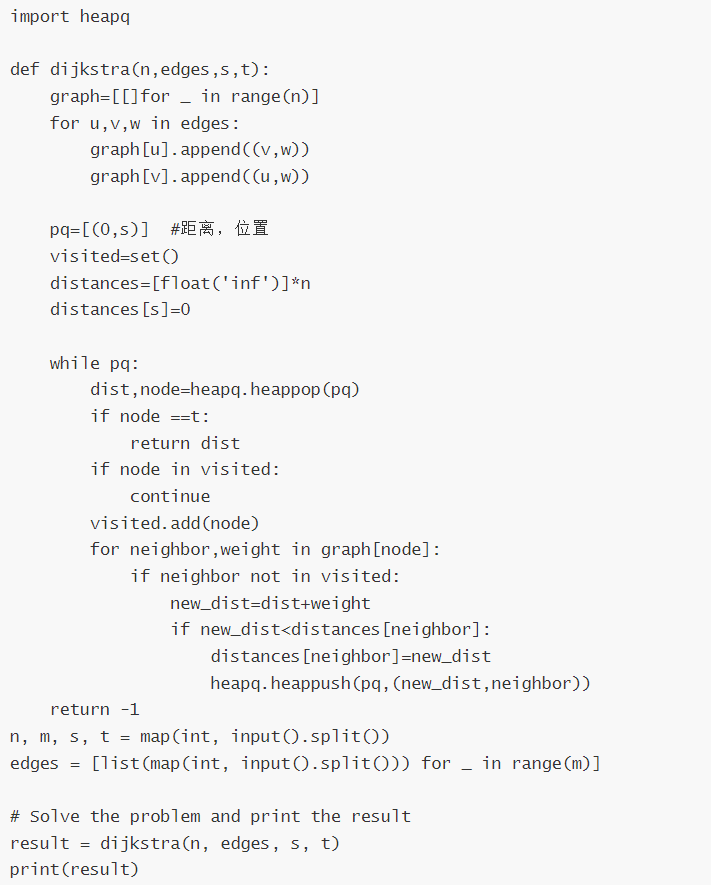
1. \*\*第一次DFS\*\*：在第一次DFS中，我们对图进行标准的深度优先搜索，但是在此过程中，我们记录下顶点完成搜索的顺序。这一步的目的是为了找出每个顶点的完成时间（即结束时间）。

2. \*\*反向图\*\*：接下来，我们对原图取反，即将所有的边方向反转，得到反向图。

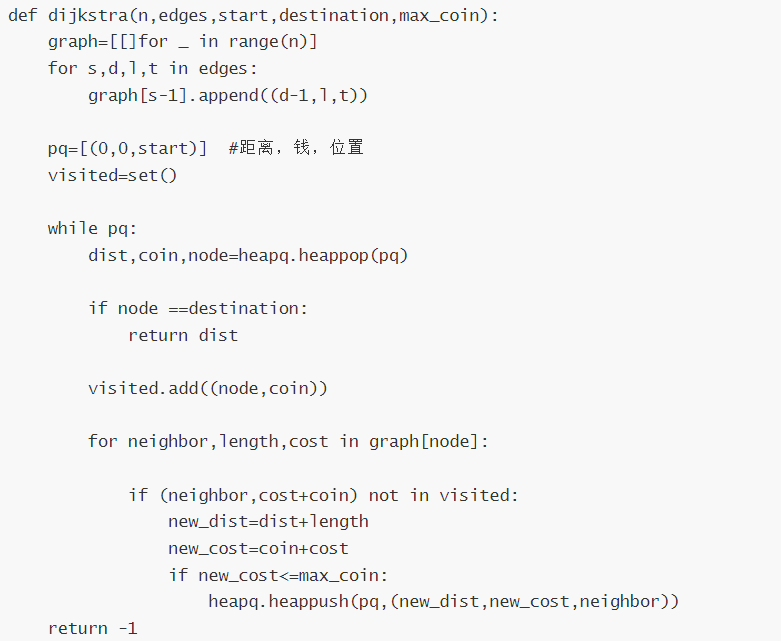
3. \*\*第二次DFS\*\*：在第二次DFS中，我们按照第一步中记录的顶点完成时间的逆序，对反向图进行DFS。这样，我们将找出反向图中的强连通分量。



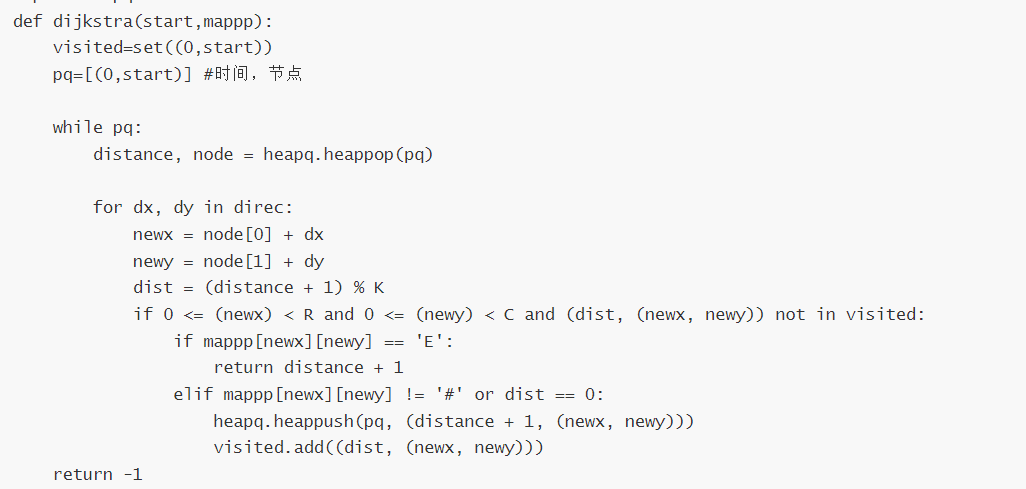
Dijkstra用到了优先队列，注意一定要把weight放在vertex前面



不能超过多少钱的最短距离：



变换迷宫：



最小生成树：MSTs \*\*如果是稠密图(边多)，则用 prim 算法;如果是稀疏图(边少)，则用 kruskal 算法\*\*。



卡车：

