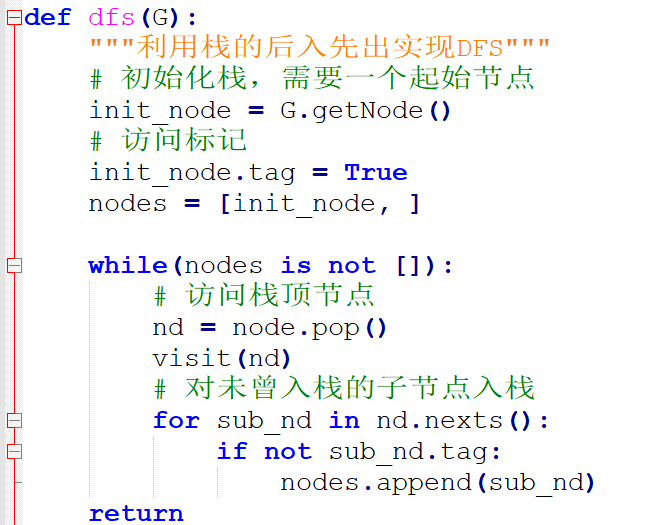
# 算法作业8

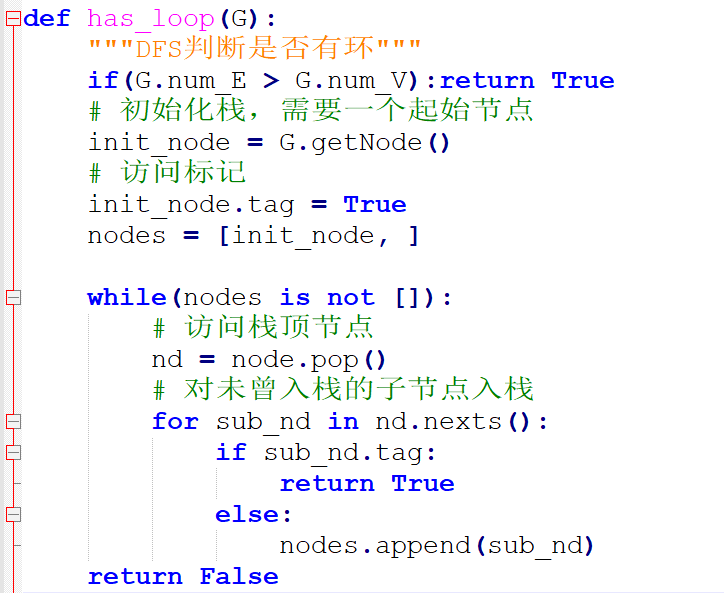
## 8.2 重写DFS算法(非递归)，利用栈

Python描述：



## 8.5设计复杂度为O(|V|)的算法确定无向图G=(V,E)是否包含回路

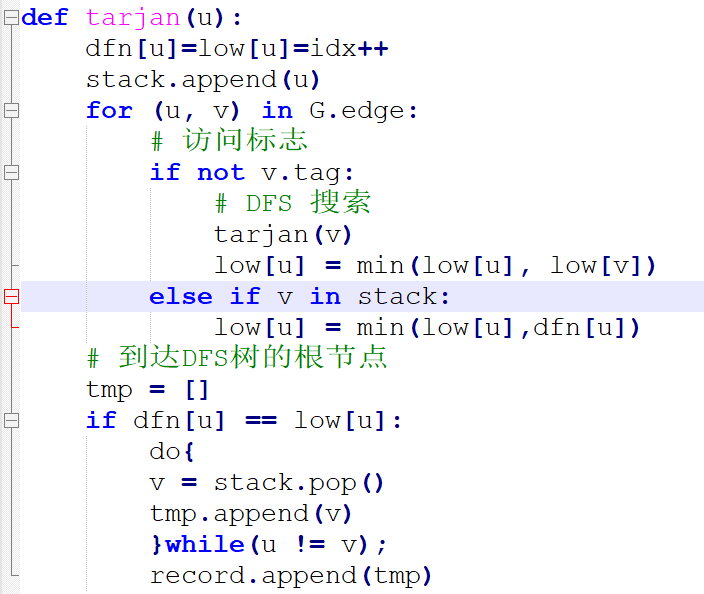
* 首先，如果无向图的边数目大于顶点数目则必存在环。若边数目小于顶点数目因此利用DFS算法(8.2)，算法复杂度T(n)=O(|v|+|E|)=O(|V|)
* 实现，利用前一题的DFS可以很快实现，如果有节点在DFS中重复访问到则必定有环



## 8.6 寻找图的连通分支

之前在数据结构课程就遇到过有向图连通分量的查找。关于有向图连通分量的寻找有个算法称为Tarjan算法，该算法基于DFS的向下搜索和回溯，对每个节点记录了两个变量low和dfn

* dfn记录这个节点在深度优先搜索中，访问的idx
* low记录了这个节点能到达的所有节点中“dfn的最小值”
* 在向下和回溯的时候不断更新low，直到遇到low=dfn的节点，这个节点就是DFS搜索树中相应连通分量的根节点
* 实现：最后的连通分量都记录在record中



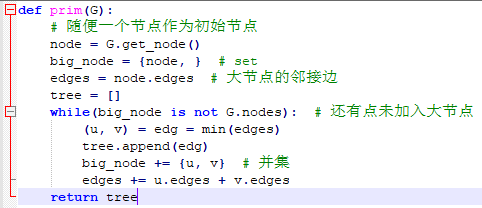
## 8.7 无向图传递闭包计算矩阵计算，要求时间复杂度

* 想到可以利用弗洛伊德算法计算，在最内层循环将路径的最小值改为逻辑运算，即有通路或者有连接，但是时间复杂度为
* 利用图的广度优先遍历算法，将每个节点作为根节点，广度优先遍历的第三层以下的子节点都可以和根节点组成传递闭包，算法复杂度为
* 花了不少时间想不到的解答，在网络上几乎都是弗洛伊德算法的改版，所以也没找到答案，分析一下复杂度的要求，只能对所有边遍历一次，就是说在一次遍历算法过程中得到解，但图的一次遍历可以看成形成一个搜索树，搜索树中非根节点间的关系会被舍弃，所有感觉的复杂度应该是不能达到的

## 8.8 设计运行时间为的Prim算法

Prim算法本身设计出来就是，应该不太需要被重新设计…，以下将重新表述一下并用伪代码描述：

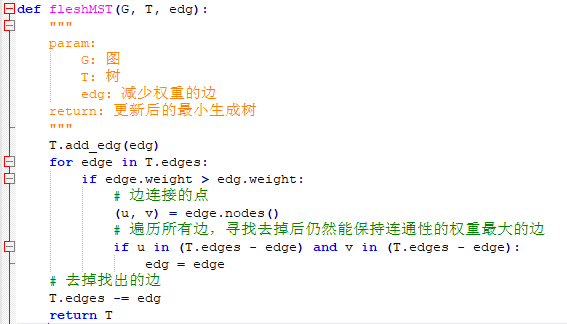
* Prim用于求解图的最小生成树问题，算法的基本步骤(循环不变式)是：将选好的点看成一个‘大节点’，在其延伸出来的边中挑出权值最小的，并将这条边连接的’小节点’加入‘大节点’
* 伪代码：



## 8.9 给定图G和最小生成树V，假设边,减少e的权重，找出修改后G的最小生成树

觉得有两种思路，根据树的存储形式不同，思路应该不一样，比如按图的形式(邻接矩阵，邻接表) 存储，则利用图可以快速访问所有边会比较简单，若按树的链表形式存储，则从头构建一个最小生成树比较快

* 在树中加入权重减小的边后，会形成一个环，若能找到环中权重最大的边，将其去掉即可，如下所示，复杂度为，因为树中边的个数比顶点个数少一个



* 若树的存储结构是链表形式，则在边改变后的图中重新运行Prim算法，代码和8.8一样，因为不断去修改树反而更加麻烦。复杂度为

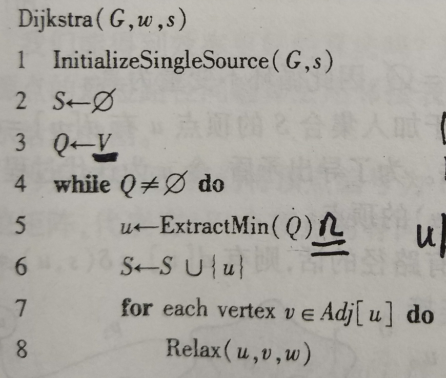
## 8.10 证明PrimMST算法的正确性

假设在某一步，Prim算法挑选出来的边集为A，参考Kruskal算法正确性证明，证明循环不变量：Prim每次往集合A中加入一条边后新的集合为最小生成树边集的子集

1. 初始步：一开始，显然为真
2. 归纳步：假设当前一步选出的边为，在加入A之前,，若不包含，则令Prim在当前步选出的点集为B，则必有，否则最小生成树为非连通的。由于Prim算法的策略，因此，在最后的生成树中将替换为得到，显然，所以也是最小生成树的边集
3. 终止步：Prim算法结束后得到的集合必为最小生成树的边集，得证

## 8.19利用合计方法分析Dijkstra算法的时间复杂度

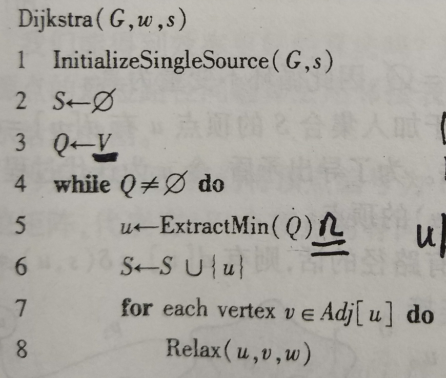
课本的Dijkstra算法



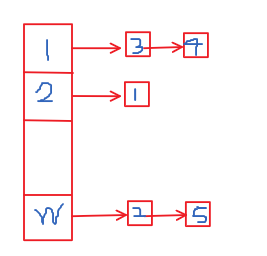
ExtractMin()操作的时间复杂度为，对每个节点都会执行一次，总的时间复杂度。而Relax()操作会对每条和当前节点邻接边进行，由于对所有节点都会执行，所以总的时间复杂度为，因此算法的时间按复杂度为，因为

## 8.22 带权有向图的权函数：，尝试修改Dijkstra算法使得复杂度为

课本的Dijkstra算法



Dijkstra算法的基本框架不用改变，要求复杂度为表明了在Relax()部分依旧得用可以随机访问的（对于洗漱图或许考虑堆）数据结构，相比之前，主要还是对ExtraMin()的优化，可以采用类似链式哈希表的结构：



存储每个权值下的邻接节点，这样Relax()仍然是遍历所有边，而ExtraMin()则可以在上面的结构中寻找最小值，复杂度为，所以算法总体复杂的为。但这里个人有个疑问，如果，那显然有，而且图为全连通图，这样，那复杂度

## 8.27 利用Floyd算法检测图的负权回路

存在负权回路的图，使用Floyd求解的时候会在负权回路上绕一圈，从而负权环上的点自身到自身的距离会是负数，即，只要检测输出矩阵的对角线，有负数存在就有负权回路，比如如下的图以及算出的输出矩阵：

