

**功能自动化测试框架实践**

**指导教师：刘冬梅**

**张静**

**516106001835**

**软件工程**

**目录**

[一、 作业要求 - 1 -](#_Toc30067)

[二、 核心算法编程实现 - 1 -](#_Toc28556)

[2.1 有向图的随机生成 - 1 -](#_Toc2915)

[2.2 回溯--深度优先搜索（DFS）算法 - 2 -](#_Toc23037)

[2.3 暴力枚举 - 3 -](#_Toc1341)

2.4 本题算法代码详细分析 - 3 -

[三、 结果和分析 - 5-](#_Toc10495)

[四、 总结 - 7-](#_Toc1005)

# 

# 作业要求

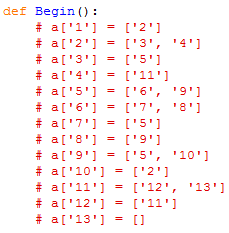
利用python编码实现：随机生成一幅有向图，指出它的首尾节点，找出它的simple paths和prime paths。

理解：simple paths相当于简单路径，内部无环，首尾节点可以相同。

Prime paths 则是最长的一条simple paths。

# 核心算法编程实现

**2.1 有向图的随机生成**

最初选用了手动输入自定义有向图的方法如下图所示：

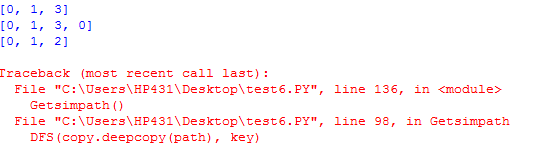
发现此方法自定义有向图输入较复杂而且对于NP问题来说，本身就是个深度递归问题，这里对栈的利用率较低。

因此考虑有向图的随机生成算法。其主要思想如下：

①利用随机邻接表，表示节点与节点之间的关系。

②随机生成6-10之间的随机数，代表有向图有6-10个节点个数。

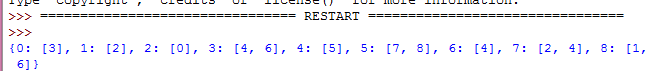
③限定每个节点的边数，randint(0,4)=1,代表设定每个点与其他点连接的概率为25%，这样避免了生成的图连接过于复杂，导致深度递归耗时长且栈溢出，如下图所示。



最终的有向图随机生成代码如下：

三次运行结果分别如下：

第一次：



第二次：

第三次：

**2.2 回溯--深度优先搜索（DFS）算法**

首先有一个概念：回溯

**回溯法**(探索与回溯法)是一种选优搜索法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。

判断两个节点间的路径。它会返回一个列表，列表里面存有组成这条路径的节点(包括起点和终点)。如果两个节点之间没有路径的话，那就返回none。相同的节点不会在返回的路径中出现两次或两次以上(就是说不会包括环)。这个算法用到了一个很重要的技术，叫做回溯：它会去尝试每一种可能，直到找到结果。不断重复，实现返回一个节点到另一个节点的所有路径，而不仅仅只查找第一条路径,从而实现了对所有路径的查找。

**深度优先算法**：

①访问初始顶点v并标记顶点v已访问。

②查找顶点v的第一个邻接顶点w。

③若顶点v的邻接顶点w存在，则继续执行；否则回溯到v，再找v的另外一个未访问过的邻接点。

④若顶点w尚未被访问，则访问顶点w并标记顶点w为已访问。

⑤继续查找顶点w的下一个邻接顶点wi，如果v取值wi转到步骤③。直到连通图中所有顶点全部访问过为止。

简单来说，就是利用栈保存未被检测的节点，节点按照深度优先的次序访问并依次被压入栈中，并以相反的次序出栈进行检测。类似于树的先根遍历。

**2.3 暴力枚举检验**

该方法即在一定范围内，对每一个可能的目标解进行检测，找出所有的目标解。枚举也称作暴力求解，穷举法。

暴力枚举的特征：

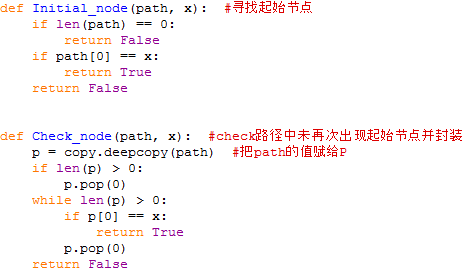
①对所有可能的目标解，逐一进行检测。

②枚举思想很简单，没有太大的思维量。（只要不超时，不超内存空间，枚举都是一种很好的方法）

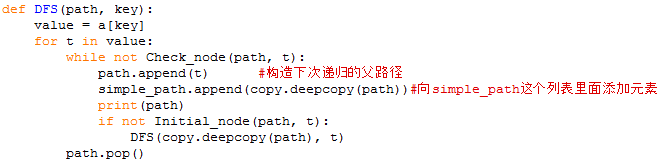
**2.4 本题算法代码详细分析**

（1）寻找初始节点

（2）根据简单路径的定义，要求内部无环，检测并确认起始节点在不在路径内部重复出现，并对它进行封装。具体代码实现如下：

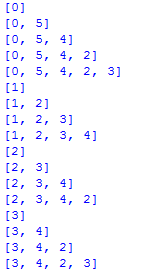
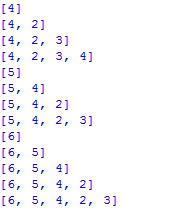


1. DFS深度优先搜索+暴力枚举 暴力检测并枚举出所有符合要求的简单路径（内部无环），具体实现如下：

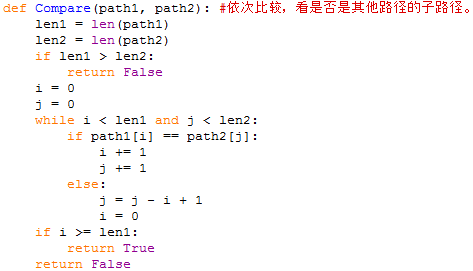


生成图：





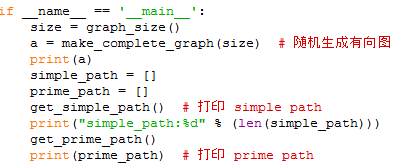
1. 拿其中一条路径与其他路径进行比较，若是其他路径的子路径则不是最长的一条简单路径，只有最长的一条简单路径才是prime path。



# 结果分析

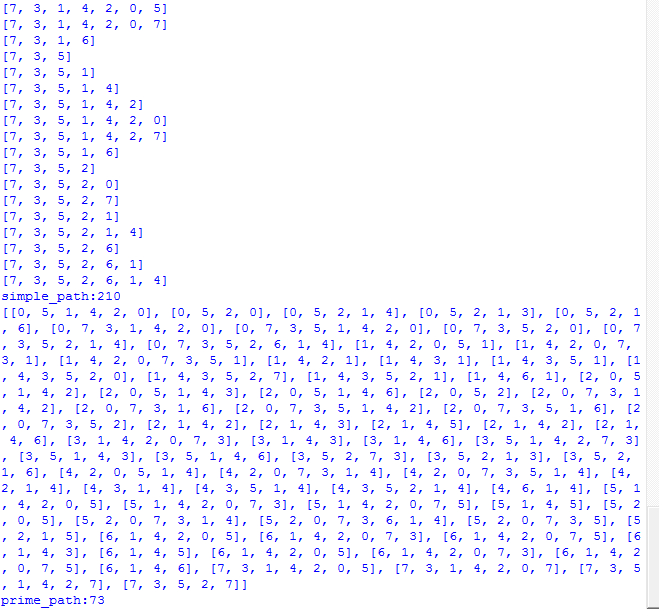
经过上面的步骤之后，我们便可以罗列出所有的simple paths 和 prime paths。具体代码和结果如下：



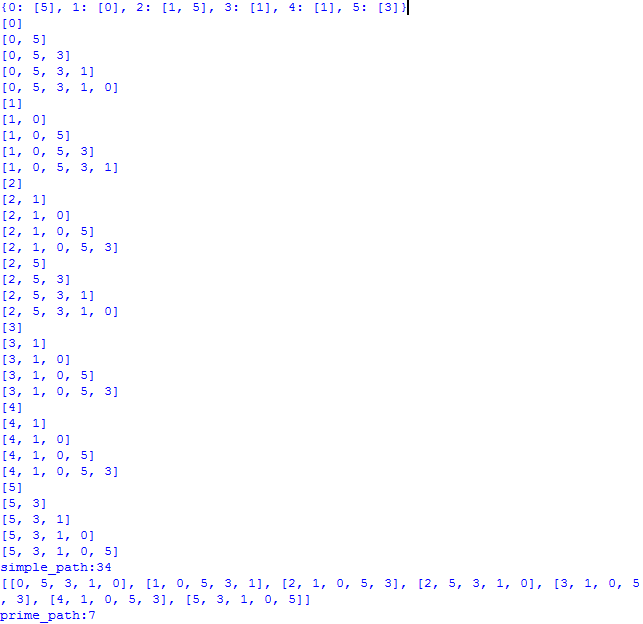


两次运行部分结果截图如下。

第一次：



第二次：



# 总结

在此次课程上，我学会了一种新的测试用例的生成方法，有时候不必自己慢慢设计单循环、二循环、多循环用例，通过寻找图的prime paths就能够实现测试用例的全覆盖，这对于设计测试用例提供很大的帮助和便捷。