*#coding=utf-8*

**class** Gragh():

**def** \_\_init\_\_(self,nodes,sides):

*'''*

*nodes 表示点*

*sides 表示边*

*'''*

*# self.sequense是字典，key是点，value是与key相连接的点*

self.sequense = {}

*# self.side是临时变量，主要用于保存与指定点相连接的点*

self.side=[]

**for** node **in** nodes:

**for** side **in** sides:

u,v=side

*# 指定点与另一个点在同一个边中，则说明这个点与指定点是相连接的点，则需要将这个点放到self.side中*

**if** node ==u:

self.side.append(v)

**elif** node == v:

self.side.append(u)

self.sequense[node] = self.side

self.side=[]

*#print self.sequense*

**'''**

**# Depth-First-Search**

**深度优先算法，是一种用于遍历或搜索树或图的算法。沿着树的深度遍历树的节点，尽可能深的搜索树的分支。**

**当节点v的所在边都己被探寻过，搜索将回溯到发现节点v的那条边的起始节点。**

**这一过程一直进行到已发现从源节点可达的所有节点为止。如果还存在未被发现的节点，**

**则选择其中一个作为源节点并重复以上过程，整个进程反复进行直到所有节点都被访问为止。属于盲目搜索。**

**'''**

**def** DFS(self,node0):

*#queue本质上是堆栈，用来存放需要进行遍历的数据*

*#order里面存放的是具体的访问路径*

queue,order=[],[]

*#首先将初始遍历的节点放到queue中，表示将要从这个点开始遍历*

queue.append(node0)

**while** queue:

*#从queue中pop出点v，然后从v点开始遍历了，所以可以将这个点pop出，然后将其放入order中*

*#这里才是最有用的地方，pop（）表示弹出栈顶，由于下面的for循环不断的访问子节点，并将子节点压入堆栈，*

*#也就保证了每次的栈顶弹出的顺序是下面的节点*

v = queue.pop()

order.append(v)

*#这里开始遍历v的子节点*

**for** w **in** self.sequense[v]:

*#w既不属于queue也不属于order，意味着这个点没被访问过，所以讲起放到queue中，然后后续进行访问*

**if** w **not in** order **and** w **not in** queue:

queue.append(w)

**return** order

**'''**

**readth-First-Search**

**BFS是从根节点开始，沿着树的宽度遍历树的节点。如果所有节点均被访问，则算法中止。**

**广度优先搜索的实现一般采用open-closed表。**

**'''**

**def** BFS(self,node0):

*#queue本质上是队列，即open表，用来存放需要进行遍历的数据*

*#order里面存放的是具体的访问路径，即closed表*

queue,order = [],[]

*#首先将初始遍历的节点放到queue中，表示将要从这个点开始遍历*

*# 由于是广度优先，也就是先访问初始节点的所有的子节点，所以可以*

queue.append(node0)

order.append(node0)

**while** queue:

*#queue.pop(0)意味着是队列的方式出元素，就是先进先出，而下面的for循环将节点v的所有子节点*

*#放到queue中，所以queue.pop(0)就实现了每次访问都是先将元素的子节点访问完毕，而不是优先叶子节点*

v = queue.pop(0)

**for** w **in** self.sequense[v]:

**if** w **not in** order:

*# 这里可以直接order.append(w) 因为广度优先就是先访问节点的所有下级子节点，所以可以*

*# 将self.sequense[v]的值直接全部先给到order*

order.append(w)

queue.append(w)

**return** order

**def** main():

*#定义节点并存储于数组nodes中*

nodes = [i+1 **for** i **in** range(8)]

*#定义边并存储于数组sides中*

sides=[(1, 2),

(1, 3),

(2, 4),

(2, 5),

(4, 8),

(5, 8),

(3, 6),

(3, 7),

(6, 7)]

G = Gragh(nodes,sides)

print(**"深度优先："**)

print(G.DFS(2))

print(**"宽度优先："**)

print(G.BFS(2))

*# print(G.DFS1(1))*

**if** \_\_name\_\_ == **"\_\_main\_\_"**:

main()