第一章 前言：

snap 官网：

http://snap.stanford.edu/snappy/index.html

API 文档：

http://snap.stanford.edu/snappy/doc/reference/index-ref.html

****

****

第二章 安装：

2.1. 所需环境：

linux 64 位，py2.6/py2.7　下载地址:

http://snap.stanford.edu/snappy/release/snap-4.1.0-4.1-centos6.5-x64-py2.6.tar.gz

试了一下 py3 没有安装成功,py3 下载地址：

http://snap.stanford.edu/snappy/release/beta/snap-5.0.9-64-3.0-centos6.5-x64-py3.6.tar.gz

2.2. snap 安装步骤：

下载安装包：(也可以手动下载)

`wget` http://snap.stanford.edu/snappy/release/snap-4.1.0-4.1-centos6.5-x64-py2.6.tar.gz

解压包：

tar zxvf snap-4.1.0-4.1-centos6.5-x64-py2.6.tar.gz

进入目录安装：

cd snap-4.1.0-4.1-centos6.5-x64-py2.6

python setup.py build

python setup.py install

```
(py27) [[redacted] snap-4.1.0-4.1-centos6.5-x64-py2.6]$ python setup.py build
running build
running build_py
(py27) [[redacted] snap-4.1.0-4.1-centos6.5-x64-py2.6]$ python setup.py install
running install
running build
running build_py
running install_lib
copying build/lib/snap.py -> /data5/jiy/opt/anaconda3/envs/py27/lib/python2.7/site-packages
byte-compiling /data5/jiy/opt/anaconda3/envs/py27/lib/python2.7/site-packages/snap.py to snap.pyc
running install_data
copying _snap.so -> /data5/jiy/opt/anaconda3/envs/py27/lib/python2.7/site-packages
running install_egg_info
Writing /data5/jiy/opt/anaconda3/envs/py27/lib/python2.7/site-packages/snap-4.1.0_dev_centos6.10_x64_py2.7-py2.7.egg-info
```

测试：

```
(py27) [[redacted]]$ python
Python 2.7.15 |Anaconda, Inc.| (default, May  1 2018, 23:32:55)
[GCC 7.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import snap
>>>
```

2.3. gnuplot 安装步骤：

官网：

http://www.gnuplot.info/index.html

下载安装包：

wget http://nchc.dl.sourceforge.net/project/gnuplot/gnuplot/4.4.0/gnuplot-4.4.0.tar.gz

解压安装包：

tar -zxvf gnuplot-4.4.0.tar.gz

配置安装目录

cd gnuplot-4.4.0

./configure --prefix=/data5/ykt/software/gnuplot

编译

make

安装

make install

配置环境变量，编译

创建 gnuplot.sh 文件：

sudo vim /etc/profile.d/gnuplot.sh

输入：

export GNUPLOT=/data5/ykt/software/gnuplot

export PATH=/data5/ykt/software/gnuplot/bin:$PATH

export MANPATH=/data5/ykt/software/gnuplot/share/man/man1:$MANPATH

保存退出

启用环境

source /etc/profile.d/gnuplot.sh

测试：

前言：

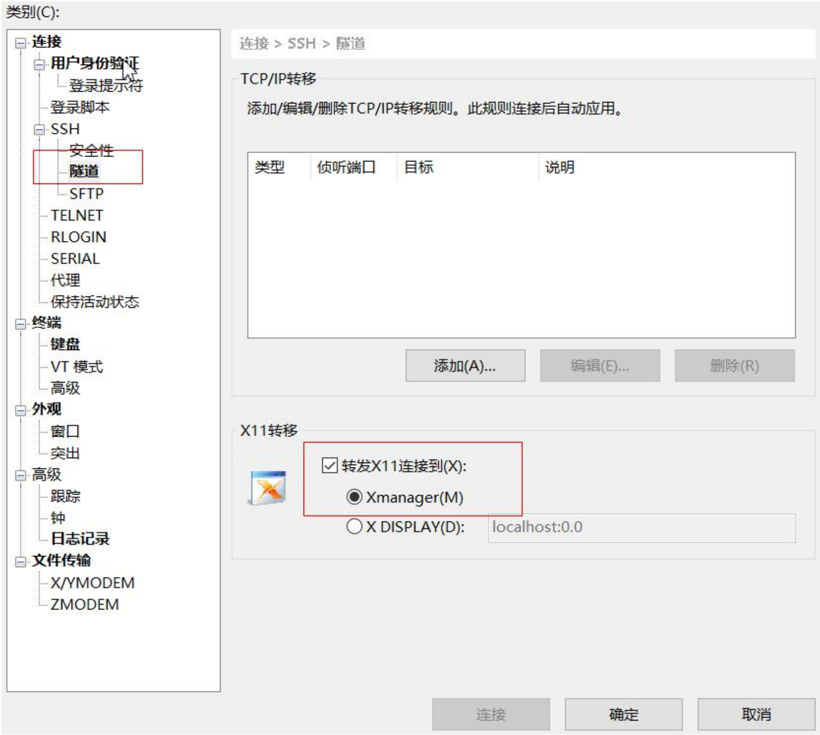
win 本地需要在 xshell 中激活 x11 功能才可以可视化，步骤如下：

先安装 Xmanager：

https://www.jb51.net/softs/624113.html

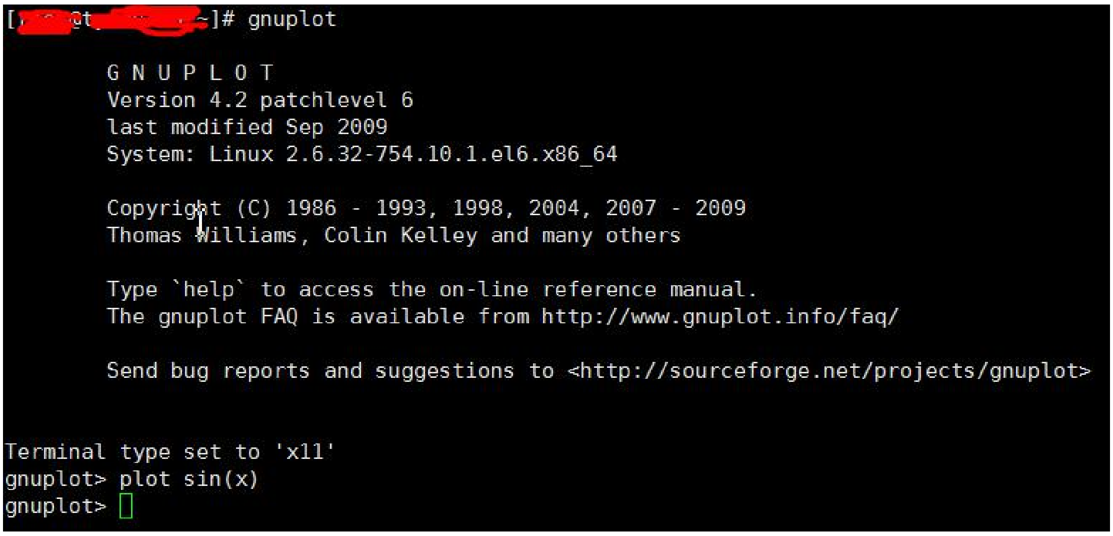
如何在 xshell 中激活 X11 转发功能：

http://www.xshellcn.com/xsh_column/x11-jih4.html



输入 gnuplot 进入其命令行

plot sin(x)



2.4. graphviz 安装

yum install graphviz

##注意：如果安装不成功，可以采取折中的办法，即在 win 上面安装 graphviz

然后每次使用 snap 会生成.dot 文件，传回到 win

在 GVEdit.exe 中打开上述生成的.dot 文件便可得到图片

****

****

第三章 教程

一：为了更好理解本部分都进行了可视化，关于 snap 的可视化部分：
http://snap.stanford.edu/snappy/doc/reference/draw.html
其中 drawgviz 是底层使用 graphviz 可视化 API

二：这里并没有讲解全部 API，只是给出了各个方面的一部分 API 作为代表，更全面的使用可以看实践部分

三: 教程和实践部分的代码大部分均 snap.py 官网给出的例子，有疑问直接去官网查，为了便于查看，下面也给出了链接。

3.1. 数据结构：

数组：snap.TIntV()

```
]:  v = snap.TIntV()
    v.Add(1)
    v.Add(2)
    v.Add(3)
    v.Add(4)
    v.Add(5)
    print(v.Len())
    print(v[2])

    5
    3
```

上述是数组的 int 类型，数组其他的类型
http://snap.stanford.edu/snappy/doc/reference/basic.html

字典： snap.TIntStrH()

```
:  h = snap.TIntStrH()

   h[5] = "five"
   h[3] = "three"
   h[9] = "nine"
   h[6] = "six"
   h[1] = "one"

   print h.Len()

   print "h[3] =", h[3]

   h[3] = "four"
   print "h[3] =", h[3]

   for key in h:
       print key, h[key]

   5
   h[3] = three
   h[3] = four
   5 five
   3 four
   9 nine
   6 six
   1 one
```

Pair 结构体：

```
h = snap.TIntStrPr(1, "one");

print h.GetVal1()
print h.GetVal2()
```

```
1
one
```

更多关于 pair ,hash 的数据结构：

http://snap.stanford.edu/snappy/doc/reference/composite.html

图(有向图/无向图),网络(带有节点和边缘属性的有向多图)：[重点]

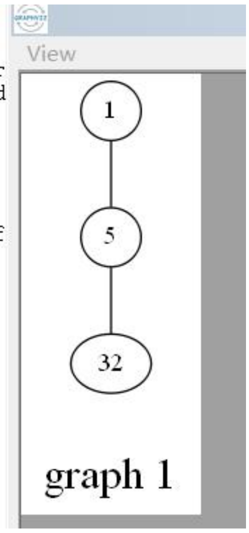
创建 3 种数据结构：

```
#无向图
G1 = snap.TUNGraph.New()
#有向图
G2 = snap.TNGraph.New()
#网络
N1 = snap.TNEANet.New()
```

给无向图 G1 添加一些节点和边并可视化：

```
In [14]: G1.AddNode(1)
         G1.AddNode(5)
         G1.AddNode(32)

         G1.AddEdge(1,5)
         G1.AddEdge(5,1)
         G1.AddEdge(5,32)

Out[14]: -1
```

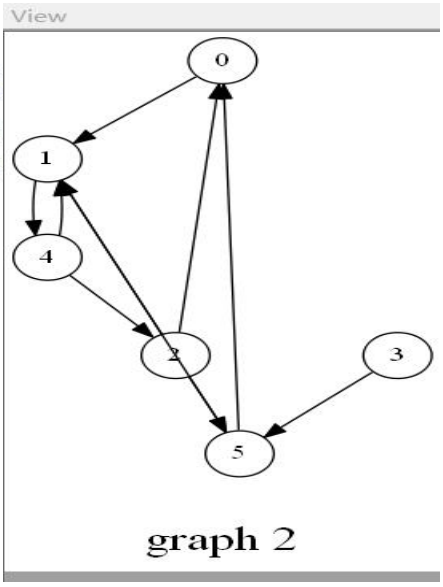
```
In [30]: snap.DrawGViz(G1, snap.gvlNeato, "graph_undirected.png", "graph 1", True)
```



有向图也可以如此添加节点和边，下面使用随机方法生成 6 个节点，10 条边的有向图

```python
# 创建一个6个节点，10条边的随机有向图
G2 = snap.GenRndGnm(snap.PNGraph, 6,10)
#打印该图的节点和边
print "G2: Nodes %d, Edges %d" % (G2.GetNodes(), G2.GetEdges())

# 遍历节点
for NI in G2.Nodes():
    print "node id %d with out-degree %d and in-degree %d" % (
        NI.GetId(), NI.GetOutDeg(), NI.GetInDeg())
# 遍历边
for EI in G2.Edges():
    print "edge (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())

# 通过节点遍历所有边
for NI in G2.Nodes():
    for Id in NI.GetOutEdges():
        print "edge (%d %d)" % (NI.GetId(), Id)

snap.DrawGViz(G2, snap.gvlNeato, "graph_undirected.png", "graph 2", True)
```

```
G2: Nodes 6, Edges 10
node id 0 with out-degree 1 and in-degree 2
node id 1 with out-degree 2 and in-degree 3
node id 2 with out-degree 3 and in-degree 1
node id 3 with out-degree 1 and in-degree 0
node id 4 with out-degree 2 and in-degree 1
node id 5 with out-degree 1 and in-degree 3
edge (0, 1)
edge (1, 4)
edge (1, 5)
edge (2, 0)
edge (2, 1)
edge (2, 5)
edge (3, 5)
edge (4, 1)
edge (4, 2)
edge (5, 0)
edge (0 1)
edge (1 4)
edge (1 5)
edge (2 0)
edge (2 1)
edge (2 5)
edge (3 5)
edge (4 1)
edge (4 2)
edge (5 0)
```



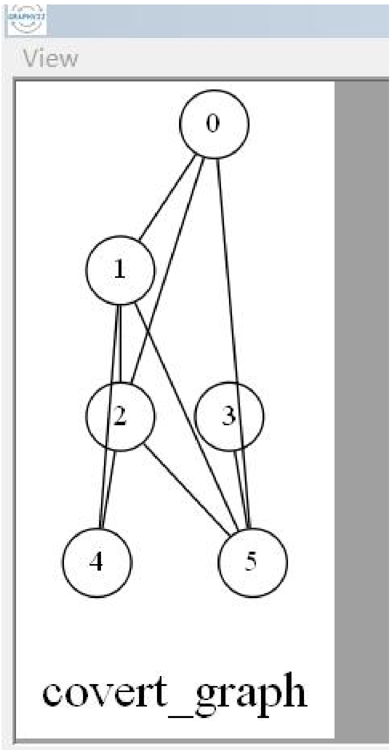
更多关于得到图属性的函数：

```
GetId(): return node id
GetOutDeg(): return out-degree of a node
GetInDeg(): return in-degree of a node
GetOutNId(e): return node id of the endpoint of e-th out-edge
GetInNId(e): return node id of the endpoint of e-th in-edge
IsOutNId(int NId): do we point to node id n
IsInNId(n): does node id n point to us
IsNbrNId(n): is node n our neighbor
```

数据转化：类如将上述有向图转化为无向图

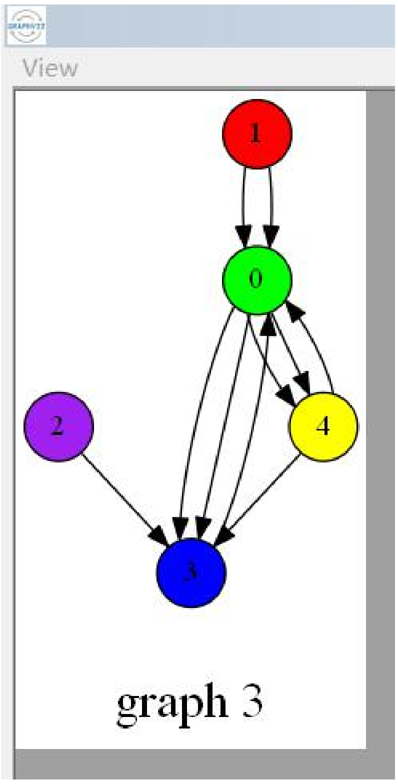
```python
#有向图转化为无向图
G3 = snap.ConvertGraph(snap.PUNGraph, G2)
snap.DrawGViz(G3, snap.gvlNeato,"covert_graph.png","covert_graph",True)
print "G3: Nodes %d, Edges %d" % (G3.GetNodes(), G3.GetEdges())
```

```
G3: Nodes 6, Edges 9
```



所谓网络就是边和节点都带属性的有向图，这里比如给节点加颜色：随机生成 5 个节点，10 条边的网络

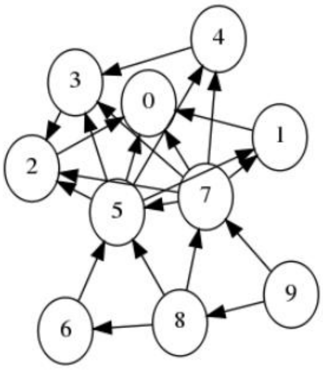
```
#节点颜色
NIdColorH = snap.TIntStrH()
NIdColorH[0] = "green"
NIdColorH[1] = "red"
NIdColorH[2] = "purple"
NIdColorH[3] = "blue"
NIdColorH[4] = "yellow"
Network = snap.GenRndGnm(snap.PNEANet, 5, 10)
snap.DrawGViz(Network, snap.gvlSfdp, "network.png", "graph 3", True, NIdColorH)
```



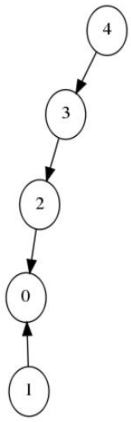
除了上述的随机生成法，还可以使用 Forest Fire model，这里同时演示了提取一个子网络的用法，可提取满足度大于等于 3 的网络

```
]: # 使用Forest Fire model产生10个节点的网络
   G4 = snap.GenForestFire(10, 0.35, 0.35)
   print "G4: Nodes %d, Edges %d" % (G4.GetNodes(), G4.GetEdges())
   snap.DrawGViz(G4,snap.gvlNeato,"Network.png","graph4",True)
   # 提取包含节点0,1, 2,3,4的子网络
   Sub_G4 = snap.GetSubGraph(G4, snap.TIntV.GetV(0,1,2,3,4))
   print "sub_G4: Nodes %d, Edges %d" % (Sub_G4.GetNodes(), Sub_G4.GetEdges())
   snap.DrawGViz(Sub_G4,snap.gvlNeato,"sub_Network.png","sub_graph4",True)

   G4: Nodes 10, Edges 21
   sub_G4: Nodes 5, Edges 4
```

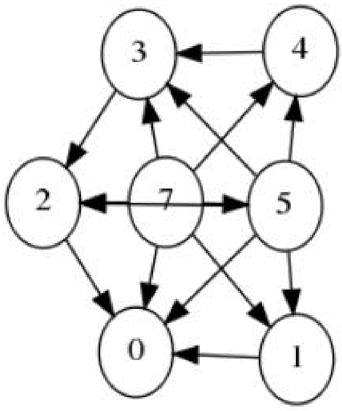
```
# get 3-core of G8
Core3 = snap.GetKCore(G4, 3)
print "Core3: Nodes %d, Edges %d" % (Core3.GetNodes(), Core3.GetEdges())
snap.DrawGViz(Core3,snap.gvlNeato,"Core3.png","Core3",True)

Core3: Nodes 7, Edges 15
```



graph4



sub_graph4



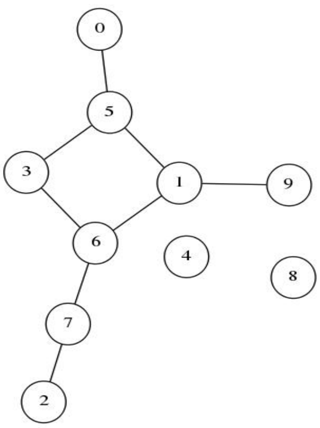
Core3

3.2.结构属性计算

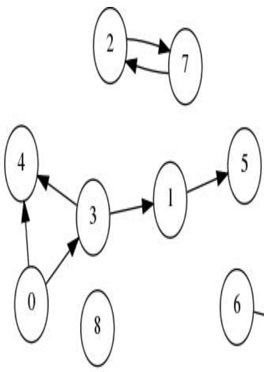
本部分仅仅剖析了每一大类下的最具代表的 API 含义，更全面的结构属性可直接看第三部分：实践。

为了全面直观的理解一些结构属性计算的函数，这里先后建立了无向图，有向图和网络，看其在上面的效果：三者都是 10 个节点，8 条边

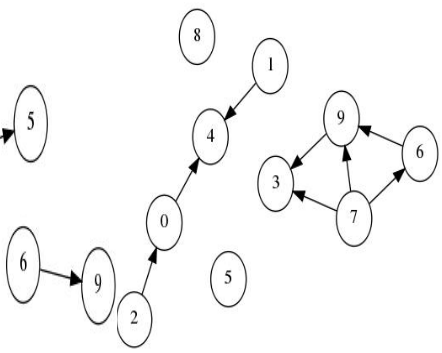
```
#无向图
UGraph = snap.GenRndGnm(snap.PUNGraph, 10,8)
snap.DrawGViz(UGraph, snap.gvlNeato, "UGraph.png", "UGraph", True)

#有向图
Graph = snap.GenRndGnm(snap.PNGraph, 10 ,8)
snap.DrawGViz(Graph, snap.gvlNeato, "Graph.png", "Graph", True)

#网络
Network = snap.GenRndGnm(snap.PNEANet, 10, 8)
snap.DrawGViz(Network, snap.gvlNeato, "Network.png", "Network", True)
```



UGraph



Graph



Network

3.2.1. 连通方面的图属性

##

GetWccSzCnt：计算的是最大弱连通分量，可以简单看成就是能够互通的单元个数，举例来说对于有向图，2 和 7 是一个单元，6 和 9 是一个单元，8 自己是一个单元，0,1,3,4,5 是一个单元，所以一共有 4 个单元，该 API 返回的结果是单元种数（以单元内节点个数为依据划分），对于上述有向图来说是 3 种，分别是包含 1 节点的单元，2 节点的单元，5 节点的单元

```
print("=============UGraph==================")
ComponentDist = snap.TIntPrV()
snap.GetWccSzCnt(UGraph, ComponentDist)
for comp in ComponentDist:
    print "Size: %d - Number of Components: %d" % (comp.GetVal1(), comp.GetVal2())

print("==============Graph==================")
ComponentDist = snap.TIntPrV()
snap.GetWccSzCnt(Graph, ComponentDist)
for comp in ComponentDist:
    print "Size: %d - Number of Components: %d" % (comp.GetVal1(), comp.GetVal2())

print("==============Network==================")
ComponentDist = snap.TIntPrV()
snap.GetWccSzCnt(Network, ComponentDist)
for comp in ComponentDist:
    print "Size: %d - Number of Components: %d" % (comp.GetVal1(), comp.GetVal2())
```

```
=============UGraph==================
Size: 1 - Number of Components: 2
Size: 8 - Number of Components: 1
==============Graph==================
Size: 1 - Number of Components: 1
Size: 2 - Number of Components: 2
Size: 5 - Number of Components: 1
==============Network==================
Size: 1 - Number of Components: 2
Size: 4 - Number of Components: 2
```

返回的 pair 第一个值代表该单元内包含的节点数，第二个值代表这种单元有多少个。

GetMxWcc：获取最大弱连通分量（WCC）的结点和边，举例对于上述有向图来说，最大的单元就是包含了 5 个节点的那个单元

```
print("=============UGraph===================")
MxWcc = snap.GetMxWcc(UGraph)
for EI in MxWcc.Edges():
    print "edge: (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())

print("=============Graph===================")
MxWcc = snap.GetMxWcc(Graph)
for EI in MxWcc.Edges():
    print "edge: (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())

print("=============Network===================")
MxWcc = snap.GetMxWcc(Network)
for EI in MxWcc.Edges():
    print "edge: (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())
```

```
=============UGraph===================
edge: (0, 5)
edge: (1, 5)
edge: (1, 6)
edge: (1, 9)
edge: (2, 7)
edge: (3, 5)
edge: (3, 6)
edge: (6, 7)
=============Graph===================
edge: (0, 3)
edge: (0, 4)
edge: (1, 5)
edge: (3, 1)
edge: (3, 4)
=============Network===================
edge: (7, 9)
edge: (7, 6)
edge: (6, 9)
edge: (7, 3)
edge: (9, 3)
```

GetMxScc：获取最大强连通分量（SCC）的结点和边，强连通：任意两个节点可以互相到达。

```
print("=============UGraph===================")
MxScc = snap.GetMxScc(UGraph)
for EI in MxScc.Edges():
    print "edge: (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())

print("=============Graph===================")
MxScc = snap.GetMxScc(Graph)
for EI in MxScc.Edges():
    print "edge: (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())

print("=============Network===================")
MxScc = snap.GetMxScc(Network)
for EI in MxScc.Edges():
    print "edge: (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())
```

```
=============UGraph===================
edge: (2, 7)
edge: (3, 5)
edge: (3, 6)
edge: (6, 7)
edge: (1, 5)
edge: (1, 6)
edge: (1, 9)
edge: (0, 5)
=============Graph===================
edge: (7, 2)
edge: (2, 7)
=============Network===================
```

GetSccSzCnt：类似 GetWccSzCnt。

更多相关连通方面的属性：

- GetSccs
- GetSccSzCnt
- GetWccs
- GetWccSzCnt
- GetMxBiCon
- GetMxScc
- GetMxSccSz
- GetMxWcc
- GetMxWccSz
- IsConnected
- IsWeaklyConn
- GetNodeWcc
- Get1CnCom
- Get1CnComSzCnt
- GetBiCon
- GetBiConSzCnt
- GetArtPoints
- GetEdgeBridges

3.2.2. 节点度的方面的属性

##

GetOutDegCnt：计算节点的出度(对于无向图就是与该节点相连的边数)，其返回的也是出度的种数，按递增的顺序给出：

```
print("==============UGraph====================")
DegToCntV = snap.TIntPrV()
snap.GetOutDegCnt(UGraph, DegToCntV)
for item in DegToCntV:
    print "%d nodes with out-degree %d" % (item.GetVal2(), item.GetVal1())

print("===============Graph====================")
DegToCntV = snap.TIntPrV()
snap.GetOutDegCnt(Graph, DegToCntV)
for item in DegToCntV:
    print "%d nodes with out-degree %d" % (item.GetVal2(), item.GetVal1())

print("==============Network===================")
DegToCntV = snap.TIntPrV()
snap.GetOutDegCnt(Network, DegToCntV)
for item in DegToCntV:
    print "%d nodes with out-degree %d" % (item.GetVal2(), item.GetVal1())
```

```
==============UGraph====================
2 nodes with out-degree 0
3 nodes with out-degree 1
2 nodes with out-degree 2
3 nodes with out-degree 3
===============Graph====================
4 nodes with out-degree 0
4 nodes with out-degree 1
2 nodes with out-degree 2
==============Network===================
4 nodes with out-degree 0
5 nodes with out-degree 1
1 nodes with out-degree 3
```

更多关于节点度方面的属性：

- CntDegNodes
- CntInDegNodes
- CntOutDegNodes
- CntNonZNodes
- GetDegCnt
- GetInDegCnt
- GetOutDegCnt
- GetMxDegNId
- GetMxInDegNId
- GetMxOutDegNId
- GetNodeInDegV
- GetNodeOutDegV
- GetDegSeqV
- GetDegSeqV

3.2.3. 图矩阵特征和奇异值方面的属性

##

GetEigVec 计算的是无向图邻接矩阵的特征向量的第一个列向量

```
print("==============UGraph====================")
EigVec =  snap.TFltV()
snap.GetEigVec(UGraph, EigVec)
for Val in EigVec:
    print Val
```

```
==============UGraph====================
-0.195300697922
-0.488611294558
-0.106263432047
-0.400569701206
6.67302672926e-06
-0.460256684403
-0.483587311186
-0.250320752672
6.67302672926e-06
-0.20740119412
```

为了验证结果这里手动将上述无向图的连接矩阵（对称矩阵）写出来然后求特征向量对比：

```
import numpy as np
x= np.array([[1,0,0,0,0,1,0,0,0,0],
             [0,1,0,0,0,1,1,0,0,1],
             [0,0,1,0,0,0,0,1,0,0],
             [0,0,0,1,0,1,1,0,0,0],
             [0,0,0,0,1,0,0,0,0,0],
             [1,1,0,1,0,1,0,0,0,0],
             [0,1,0,1,0,0,1,1,0,0],
             [0,0,1,0,0,0,1,1,0,0],
             [0,0,0,0,0,0,0,0,1,0],
             [0,1,0,0,0,0,0,0,0,1]])
a,b=np.linalg.eig(x)
print(b[:,0])
```

```
[-0.19533734 -0.48858238 -0.10624151 -0.40058017  0.          -0.46026441
 -0.48360427 -0.25033199  0.          -0.20735556]
```

该 API 只对无向图进行计算，其他会报错：

```
print("=============UGraph===================")
EigVec =  snap.TFltV()
snap.GetEigVec(Graph, EigVec)
for Val in EigVec:
    print Val
```

```
=============UGraph===================

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-3652587ff6ba> in <module>()
      1 print("=============UGraph===================")
      2 EigVec =  snap.TFltV()
----> 3 snap.GetEigVec(Graph, EigVec)
      4 for Val in EigVec:
      5     print Val

/data5/jiy/opt/anaconda3/envs/py27/lib/python2.7/site-packages/snap.pyc in GetEigVec(*args)
  35593
  35594     """
> 35595     return _snap.GetEigVec(*args)
  35596
  35597 def GetInvParticipRat(Graph, MaxEigVecs, TimeLimit, EigValIprV):

TypeError: in method 'GetEigVec', argument 1 of type 'PUNGraph const &'
```

更多关于该方面的图属性：

- GetEigVals
- GetEigVec
- GetEigVec
- GetSngVals
- GetSngVec
- GetSngVec
- GetInvParticipRat

3.2.4. 深度遍历和宽度遍历方面的属性

###

GetBfsFullDiam：利用宽度遍历算法计算最大宽度（深度），或者说给出图中任意一对顶点之间的最大距离，例如对于无向图其最大宽度是 5 即 0->5->1->6->7->2

```
print("==============UGraph with 10 nodes====================")
diam = snap.GetBfsFullDiam(UGraph, 10, False)
print diam

print("==============UGraph with 1 nodes====================")
diam = snap.GetBfsFullDiam(UGraph, 1, False)
print diam

print("==============Graph====================")
diam = snap.GetBfsFullDiam(Graph, 10, False)
print diam

print("==============Network====================")
diam = snap.GetBfsFullDiam(Network, 10, False)
print diam
```

```
==============UGraph with 10 nodes====================
5
==============UGraph with 1 nodes====================
4
==============Graph====================
3
==============Network====================
3
```

其中宽度遍历是要开始选取一个随机点的，以此进行遍历，第二个参数是使用多少个点作为随机点，可以看到当取 1 个点时，有可能就没有取到 0 这个点，故得到的是 4，所以为了精确还是取大一点好。

GetBfsEffDiam：先统计所有的可能的宽度，然后计算 90-th percentile of the distribution

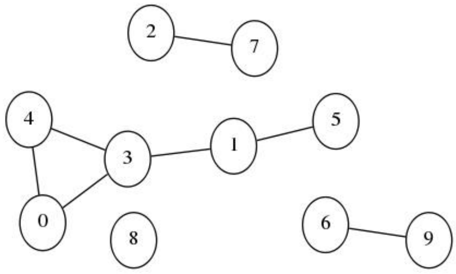
关于该方面的更多图属性有：

- GetBfsFullDiam
- GetBfsEffDiam
- GetBfsEffDiam
- GetBfsEffDiamAll
- GetNodesAtHop
- GetNodesAtHops
- GetShortPath
- GetShortPath
- GetBfsTree
- GetTreeRootNId
- GetTreeSig
- GetTreeSig

3.2.5. 三角(闭合环)和聚集系数方面的属性

##

GetTriads：首先其将所有结构看成是无向图，然后计算每一个节点的三节点闭合数量和非闭合数量，举例来说上述有向图，首先我们将其看成无向图即大概是这样子：



Graph_convert_UGraph

然后我们来随便拿一个节点来看看，比如 3 这个节点，与其相连的有 1,0,4 那么其组合的所有情况是[3,4,0],[3,0,1],[3,4,1] ,闭合的有[3,4,0]，所以对于 3 这个节点有 1 个闭合，2 个非闭合，要组成闭合，除了自身外最少还需要两个节点与自身相连，如果与其相连的只有一个节点或者没有节点与其相连，那么就不会计数，直接都为 0，如统计上图的节点 6，其闭合个数是 0，非闭合也是 0.

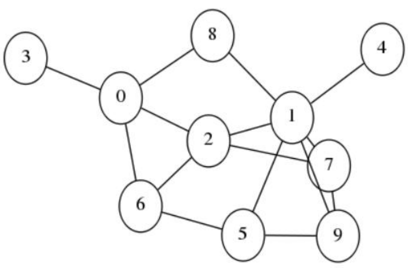
```
print("=============UGraph===================")
TriadV = snap.TIntTrV()
snap.GetTriads(UGraph, TriadV)
for triple in TriadV:
    print (triple.Val1(), triple.Val2(), triple.Val3())

print("=============Graph===================")
TriadV = snap.TIntTrV()
snap.GetTriads(Graph, TriadV)
for triple in TriadV:
    print (triple.Val1(), triple.Val2(), triple.Val3())

print("=============Network===================")
TriadV = snap.TIntTrV()
snap.GetTriads(Network, TriadV)
for triple in TriadV:
    print (triple.Val1(), triple.Val2(), triple.Val3())
```

```
==============UGraph====================
(7, 0, 1)
(8, 0, 0)
(3, 0, 1)
(9, 0, 0)
(1, 0, 3)
(0, 0, 0)
(6, 0, 3)
(2, 0, 0)
(5, 0, 3)
(4, 0, 0)
==============Graph====================
(7, 0, 0)
(8, 0, 0)
(3, 1, 2)
(9, 0, 0)
(1, 0, 1)
(0, 1, 0)
(6, 0, 0)
(2, 0, 0)
(5, 0, 0)
(4, 1, 0)
==============Network====================
(7, 1, 2)
(8, 0, 0)
(3, 1, 0)
(9, 1, 2)
(1, 0, 0)
(0, 0, 1)
(6, 1, 0)
(2, 0, 0)
(5, 0, 0)
(4, 0, 1)
```

返回的第一个数是节点 id，第二数代表闭合的数量，第三个数是非闭合数量，同时通过上面的分析我们可以得到，假设与当前节点相连的有 N（N>=2）节点，那么其所有组合的情况就是 C_N^2,所以返回的第二数和第三个的和一定等于 C_N^2

为了便于进一步验证结果，这里通过多加边（15 条边）产生更多闭合环来看看：

```
#无向图_2
UGraph_2 = snap.GenRndGnm(snap.PUNGraph, 10,15)
snap.DrawGViz(UGraph_2,snap.gvlNeato,"UGraph_2.png","UGraph_2",True)
```



UGraph_2

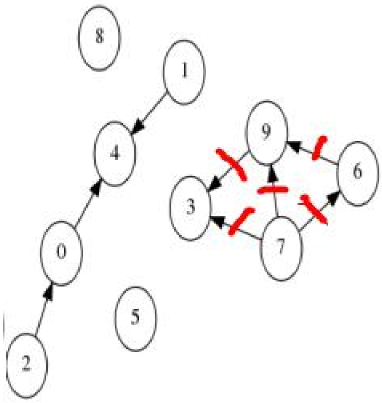
```
print("==============UGraph_2====================")
TriadV = snap.TIntTrV()
snap.GetTriads(UGraph_2, TriadV)
for triple in TriadV:
    print (triple.Val1(), triple.Val2(), triple.Val3())
```

```
==============UGraph_2====================
(7, 2, 1)
(8, 0, 1)
(3, 0, 0)
(9, 2, 1)
(1, 3, 12)
(0, 1, 5)
(6, 1, 2)
(2, 2, 4)
(5, 1, 2)
(4, 0, 0)
```

就节点 1 而言，与其相连的有 4,8,2,7,5,9 即 6 个节点，一共是 15 种组合情况，闭合的有[1,7,9]

,[1,5,9],[1,2,7]即 3 种情况，其余 12 种均为非闭合情况。

GetTriadEdges：计算参与闭合边的条数，举例上述的网络图：还是先转化为无向图，然后看看有多少边参与了闭合即 5 条：




```
print("=============UGraph===================")
NumTriadEdges = snap.GetTriadEdges(UGraph)
print (NumTriadEdges)

print("=============Graph===================")
NumTriadEdges = snap.GetTriadEdges(Graph)
print (NumTriadEdges)


print("=============Network===================")
NumTriadEdges = snap.GetTriadEdges(Network)
print (NumTriadEdges)


print("=============UGraph_2===================")
NumTriadEdges = snap.GetTriadEdges(UGraph_2)
print (NumTriadEdges)
```

```
=============UGraph===================
0
=============Graph===================
3
=============Network===================
5
=============UGraph_2===================
10
```

GetTriadParticip：遍历每个节点，计算其参与闭合环的次数，举例对于上述网络图,节点 7,9 都参与了 2 次，节点 3,6 各参与了一次，其余节点参与了 0 次，所以参与了 0 次的有 6 个节点，参与了 1 次的有两个节点，参与了 2 次的有 2 个节点。

```
print("=============UGraph===================")
TriadCntV = snap.TIntPrV()
snap.GetTriadParticip(UGraph, TriadCntV)
for pair in TriadCntV:
    print pair.Val1(), pair.Val2()

print("=============Graph===================")
TriadCntV = snap.TIntPrV()
snap.GetTriadParticip(Graph, TriadCntV)
for pair in TriadCntV:
    print pair.Val1(), pair.Val2()


print("=============Network===================")
TriadCntV = snap.TIntPrV()
snap.GetTriadParticip(Network, TriadCntV)
for pair in TriadCntV:
    print pair.Val1(), pair.Val2()


print("=============UGraph_2===================")
TriadCntV = snap.TIntPrV()
snap.GetTriadParticip(UGraph_2, TriadCntV)
for pair in TriadCntV:
    print pair.Val1(), pair.Val2()
```

```
=============UGraph===================
0 10
=============Graph===================
0 7
1 3
=============Network===================
0 6
1 2
2 2
=============UGraph_2===================
0 3
1 3
2 3
3 1
```

GetClustCf：聚集系数，其分为全局和部分计算如下：

全局计算是：闭合环/非闭合环

部分计算：局部计算是面向节点的，对于节点 vi，找出其直接邻居节点集合 Ni，计算 Ni 构成的网络中的边数 K，除以 Ni 集合可能的边数|Ni|*（|Ni|-1）/2 更多细节可以看：

https://blog.csdn.net/pennyliang/article/details/6838956

```
print("==============UGraph====================")
CfVec = snap.TFltPrV()
Cf = snap.GetClustCf(UGraph, CfVec, -1)
print(Cf)
#print "Average Clustering Coefficient: %f" % (Cf)
print "Coefficients by degree:\n"
for pair in CfVec:
    print "degree: %d, clustering coefficient: %f" % (pair.GetVal1(), pair.GetVal2())

print("==============Graph====================")
CfVec = snap.TFltPrV()
Cf = snap.GetClustCf(Graph, CfVec, -1)
print(Cf)
#print "Average Clustering Coefficient: %f" % (Cf)
print "Coefficients by degree:\n"
for pair in CfVec:
    print "degree: %d, clustering coefficient: %f" % (pair.GetVal1(), pair.GetVal2())

print("==============Network====================")
CfVec = snap.TFltPrV()
Cf = snap.GetClustCf(Network, CfVec, -1)
print(Cf)
#print "Average Clustering Coefficient: %f" % (Cf)
print "Coefficients by degree:\n"
for pair in CfVec:
    print "degree: %d, clustering coefficient: %f" % (pair.GetVal1(), pair.GetVal2())
```

```
==============UGraph====================
[0.0, 0, 11]
Coefficients by degree:

degree: 0, clustering coefficient: 0.000000
degree: 1, clustering coefficient: 0.000000
degree: 2, clustering coefficient: 0.000000
degree: 3, clustering coefficient: 0.000000
==============Graph====================
[0.23333333333333333, 1, 3]
Coefficients by degree:

degree: 0, clustering coefficient: 0.000000
degree: 1, clustering coefficient: 0.000000
degree: 2, clustering coefficient: 0.400000
degree: 3, clustering coefficient: 0.333333
==============Network====================
[0.26666666666666666, 1, 6]
Coefficients by degree:

degree: 0, clustering coefficient: 0.000000
degree: 1, clustering coefficient: 0.000000
degree: 2, clustering coefficient: 0.500000
degree: 3, clustering coefficient: 0.333333
```

其中 Cf 的列表的第一个数就是全局聚集系数，对于局部聚集系数以网络的度为 2 的节点举例，其度为 2 的节点有 0,4,3,6 其聚集系数分别为 0,0,1,1，所以该 4 个节点的平均聚集系数是(1+1)/4 = 0.5

GetClustCfAll:同上

```
print("=============UGraph===================")
DegToCCfV = snap.TFltPrV()
result = snap.GetClustCfAll(UGraph, DegToCCfV)
for item in DegToCCfV:
    print "degree: %d, clustering coefficient: %f" % (item.GetVal1(), item.GetVal2())
print "average clustering coefficient", result[0]
print "closed triads", result[1]
print "open triads", result[2]

print("=============Graph===================")
DegToCCfV = snap.TFltPrV()
result = snap.GetClustCfAll(Graph, DegToCCfV)
for item in DegToCCfV:
    print "degree: %d, clustering coefficient: %f" % (item.GetVal1(), item.GetVal2())
print "average clustering coefficient", result[0]
print "closed triads", result[1]
print "open triads", result[2]

print("=============Network===================")
DegToCCfV = snap.TFltPrV()
result = snap.GetClustCfAll(Network, DegToCCfV)
for item in DegToCCfV:
    print "degree: %d, clustering coefficient: %f" % (item.GetVal1(), item.GetVal2())
print "average clustering coefficient", result[0]
print "closed triads", result[1]
print "open triads", result[2]
```

```
==============UGraph====================
degree: 0, clustering coefficient: 0.000000
degree: 1, clustering coefficient: 0.000000
degree: 2, clustering coefficient: 0.000000
degree: 3, clustering coefficient: 0.000000
average clustering coefficient 0.0
closed triads 0
open triads 11
==============Graph====================
degree: 0, clustering coefficient: 0.000000
degree: 1, clustering coefficient: 0.000000
degree: 2, clustering coefficient: 0.400000
degree: 3, clustering coefficient: 0.333333
average clustering coefficient 0.233333333333
closed triads 1
open triads 3
==============Network====================
degree: 0, clustering coefficient: 0.000000
degree: 1, clustering coefficient: 0.000000
degree: 2, clustering coefficient: 0.500000
degree: 3, clustering coefficient: 0.333333
average clustering coefficient 0.266666666667
closed triads 1
open triads 6
```

两者的区别在于使用 GetClustCf 时，当不传入 Cfvec 时，返回的就只是全局聚集系数：

```
print("=============UGraph===================")
Cf = snap.GetClustCf(UGraph)
print "Average Clustering Coefficient: %f" % (Cf)

print("=============Graph===================")
CfVec = snap.TFltPrV()
Cf = snap.GetClustCf(Graph)
print "Average Clustering Coefficient: %f" % (Cf)

print("=============Network===================")
CfVec = snap.TFltPrV()
Cf = snap.GetClustCf(Network)
print "Average Clustering Coefficient: %f" % (Cf)
```

```
=============UGraph===================
Average Clustering Coefficient: 0.000000
=============Graph===================
Average Clustering Coefficient: 0.233333
=============Network===================
Average Clustering Coefficient: 0.266667
```

关于该方面更多图属性：

- GetClustCf
- GetClustCf
- GetClustCfAll
- GetTriads
- GetTriads
- GetTriadsAll
- GetCmnNbrs
- GetCmnNbrs
- GetNodeClustCf
- GetNodeClustCf
- GetNodeTriads
- GetNodeTriads
- GetNodeTriadsAll
- GetLen2Path
- GetLen2Paths
- GetTriadEdges
- GetTriadParticip

更多关于计算图结构属性的 API 可以到其官方 API 文档中查看：

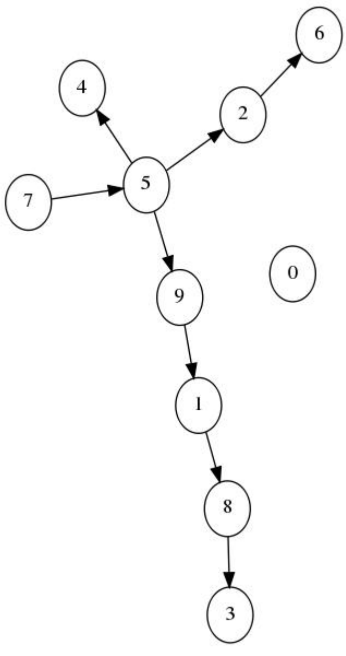
- Introduction
- Basic Types
- Composite Types
- File Streams
- Graph and Network Classes
- Tables
- Multimodal Networks
- Sparse Attributes
- Graph Generators
- Table to Graph Conversion Functions
- Input and Output
- Node Degree
- Edge Count
- Graph Manipulation
- Subgraphs and Graph Type Conversions
- Graph Information
- Plotting and Drawing
- Connected Components
- Breadth and Depth First Search
- Node Centrality
- Community Detection
- Triads and Clustering Coefficient
- K-core
- Approximate Neighborhood
- Eigen and Singular Value Decomposition
- Contributors

3.3. 文件流(数据的保存和加载)：

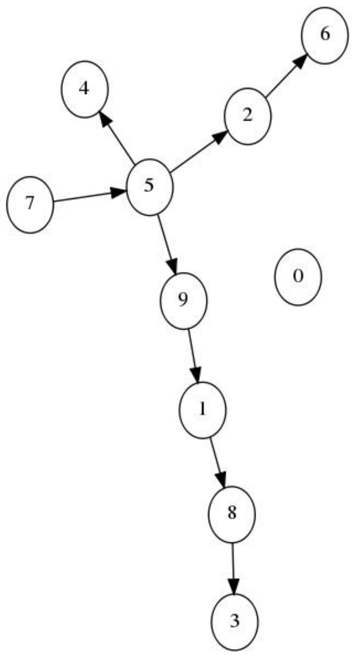
二进制文件

```
#有向图
Graph = snap.GenRndGnm(snap.PNGraph, 10 ,8)
snap.DrawGViz(Graph, snap.gvlNeato, "Graph.png", "Graph", True)
```

```
#二进制保存和加载有向图
FOut = snap.TFOut("Graph.graph")
Graph.Save(FOut)
FOut.Flush()
FIn = snap.TFIn("Graph.graph")
Graph_load = snap.TNGraph.Load(FIn)
snap.DrawGViz(Graph_load, snap.gvlNeato, "Graph_load.png", "Graph_load", True)
```



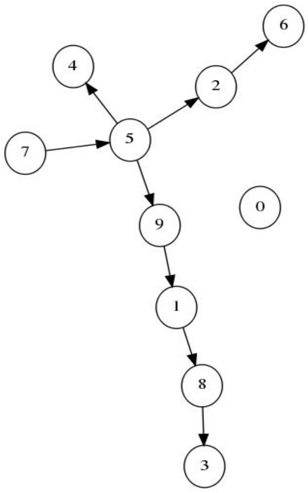
Graph



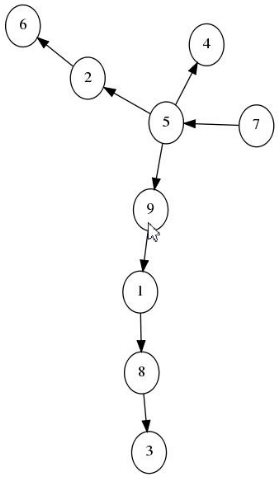
Graph_load

通过 txt 文件

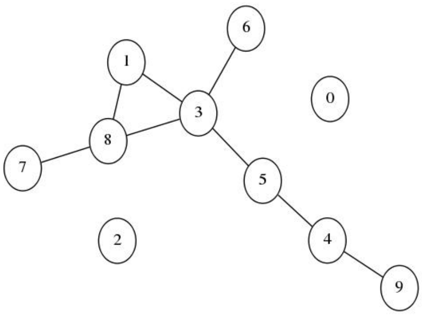
```
snap.SaveEdgeList(UGraph, "UGraph.txt", "Save UGraph as tab-separated list of edges")
UGraph_load = snap.LoadEdgeList(snap.PUNGraph, "UGraph.txt", 0, 1)
snap.DrawGViz(UGraph_load, snap.gvlNeato, "UGraph_load.png", "UGraph_load", True)

snap.SaveEdgeList(Graph, "Graph.txt", "Save Graph as tab-separated list of edges")
Graph_load = snap.LoadEdgeList(snap.PNGraph, "Graph.txt", 0, 1)
snap.DrawGViz(Graph_load, snap.gvlNeato, "Graph_load.png", "Graph_load", True)

snap.SaveEdgeList(Network, "Network.txt", "Save Network as tab-separated list of edges")
Network_load = snap.LoadEdgeList(snap.PNEANet, "Network.txt", 0, 1)
snap.DrawGViz(Network_load, snap.gvlNeato, "Network_load.png", "Network_load", True)
```



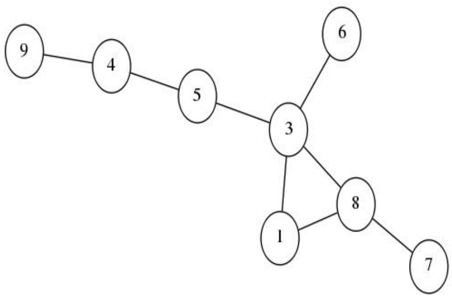
Graph



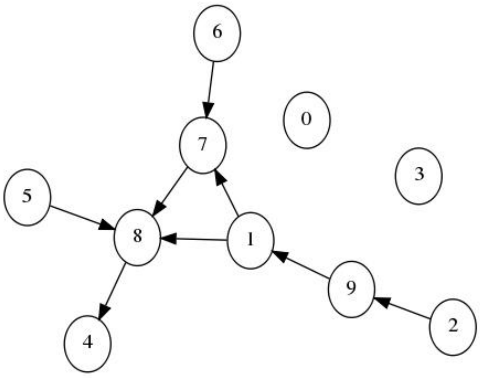
Graph_load



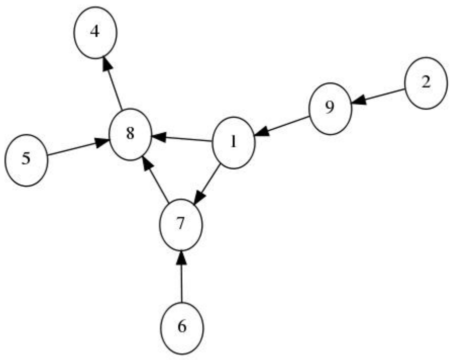
UGraph



UGraph_load



Network



Network_load

上述加载的 txt 文件形式是[source node id,destination node id]

还可以是：[source node id,destination node name 1,destination node name 2,..........]

其对应的 API 是：

LoadConnList 和 LoadConnListStr，除此之外还可以加载和保存.paj 文件具体更多的可以看：

http://snap.stanford.edu/snappy/doc/reference/io.html

****

****

第四章 实践

一：该部分实践了 2 种数据集(有向数据集/无向数据集)，均给出了 API 的具体使用和运行时间的情况

二：部分 API 是返回的结果过大，如一个很大的数组(计算所有节点或所有边的一些指标)，，所以不方便打印输出，这里就仅仅输出看一下结果的长度

4.1. 有向数据集

LiveJournal(http://snap.stanford.edu/data/soc-LiveJournal1.html)
其解压后是一个 txt 文件，内部数据结构是：(源节点->目的节点)

```
3734    888414
3734    898649
3734    899418
3734    904810
3734    957306
3734    970327
3734    977872
3734    991223
3734    1006116
3734    1049631
3734    1098339
3734    1101880
3734    1103862
3734    1108648
3734    1166751
3734    1186319
3734    1214191
3734    1216998
3734    1235479
3734    1265204
3734    1268815
3734    1448733
3734    1728708
3734    1823725
3734    1823917
3734    1917090
3734    2056273
3734    2337875
3734    2612931
```

加载数据集

```
%%time
graph = snap.LoadEdgeList(snap.PNGraph, "soc-LiveJournal1.txt", 0, 1)
```

```
CPU times: user 2min 14s, sys: 5.41 s, total: 2min 20s
Wall time: 2min 20s
```

4.1.1. 图的基本常见的统计信息

可以使用 print 函数，其会自动计算一些基本的统计信息

```
%%time
snap.PrintInfo(graph, "Python type PNGraph", "info-pngraph.txt", False)
```

```
CPU times: user 1h 6min 21s, sys: 3.19 s, total: 1h 6min 24s
Wall time: 1h 6min 25s
```

jupyter info-pngraph.txt 4 分钟前

File Edit View Language

```
Python type PNGraph: Directed
  Nodes:                        4847571
  Edges:                        68993773
  Zero Deg Nodes:               0
  Zero InDeg Nodes:             358331
  Zero OutDeg Nodes:            539119
  NonZero In-Out Deg Nodes:     3950121
  Unique directed edges:        68993773
  Unique undirected edges:      43369619
  Self Edges:                   518382
  BiDir Edges:                  51766690
  Closed triangles:             285730264
  Open triangles:               6412312961
  Frac. of closed triads:       0.042659
  Connected component size:     0.999254
  Strong conn. comp. size:      0.789815
  Approx. full diameter:        14
  90% effective diameter:       6.286255
```

=====================以下是更多指标详细的统计过程==========================

4.1.2. 度节点方面的属性

该部分 API：http://snap.stanford.edu/snappy/doc/reference/degree.html

节点数

```
%%time
print(graph.GetNodes())
```

```
4847571
CPU times: user 6 ms, sys: 5 ms, total: 11 ms
Wall time: 4.8 ms
```

边数

```
%%time
print(graph.GetEdges())
```

```
68993773
CPU times: user 799 ms, sys: 3 ms, total: 802 ms
Wall time: 798 ms
```

度为 100 的节点个数

```
%%time
#度为100的节点个数
print(snap.CntDegNodes(graph, 100))
```

```
4442
CPU times: user 1.23 s, sys: 7 ms, total: 1.23 s
Wall time: 1.23 s
```

出度为 100 的节点个数

```
%%time
#出度为100的节点个数
print(snap.CntOutDegNodes(graph, 100))
```

```
1785
CPU times: user 1.46 s, sys: 2 ms, total: 1.46 s
Wall time: 1.46 s
```

入度为 100 的节点个数

```
%%time
#入度为100的节点个数
print(snap.CntInDegNodes(graph, 100))
```

```
1809
CPU times: user 1.37 s, sys: 3 ms, total: 1.37 s
Wall time: 1.37 s
```

有着最大出度的节点的 id

```
%%time
#有着最大出度的节点的id
print(snap.GetMxOutDegNId(graph))
```

```
10009
CPU times: user 1.18 s, sys: 1 ms, total: 1.18 s
Wall time: 1.18 s
```

有着最大入度的节点的 id

```
%%time
#有着最大入度的节点的id
print(snap.GetMxInDegNId(graph))
```

```
10029
CPU times: user 1.07 s, sys: 4 ms, total: 1.07 s
Wall time: 1.07 s
```

4.1.3. 图连通方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/cncom.html

弱连通网络的个数:

```
%%time
#Components是弱连通网络集合
Components = snap.TCnComV()
snap.GetWccs(graph, Components)
print(Components.Len())
```

```
1876
CPU times: user 20.8 s, sys: 1e+03 µs, total: 20.8 s
Wall time: 20.8 s
```

弱连通网络的种数:

```
%%time
#ComponentDist相当于对Components按<弱连通网络中节点数大小>进行了分组
ComponentDist = snap.TIntPrV()
snap.GetWccSzCnt(graph, ComponentDist)
print(ComponentDist.Len())
```

```
24
CPU times: user 18.1 s, sys: 6 ms, total: 18.1 s
Wall time: 18.1 s
```

和节点 3 在同一个弱连通网路的节点集合

```
%%time
#和节点3在同一个弱连通网路的节点集合
CnCom = snap.TIntV()
snap.GetNodeWcc(graph, 3, CnCom)
# print "Nodes in the same connected component as node 0:"
# for node in CnCom:
#     print node
# print('--')
print(CnCom.Len())
```

```
4843953
CPU times: user 14.7 s, sys: 5 ms, total: 14.7 s
Wall time: 14.7 s
```

最大弱连通网络的节点数和边数

```
%%time
MxWcc = snap.GetMxWcc(graph)
print(MxWcc.GetNodes())
print(MxWcc.GetEdges())
```

```
4843953
68983820
CPU times: user 1min 42s, sys: 1.63 s, total: 1min 43s
Wall time: 1min 44s
```

、

最大弱连通网络的节点分数

```
%%time
print(snap.GetMxWccSz(graph))
```

```
0.999253646826
CPU times: user 31.3 s, sys: 2 ms, total: 31.3 s
Wall time: 31.4 s
```

强连通网络的个数：

```
%%time
#Components是强连通网络集合
Components = snap.TCnComV()
snap.GetSccs(graph, Components)
print(Components.Len())
```

```
971232
CPU times: user 47.6 s, sys: 13 ms, total: 47.6 s
Wall time: 47.6 s
```

强连通网络的种数：

```
%%time
#ComponentDist相当于对Components按<强连通网络中节点数大小>进行了分组
ComponentDist = snap.TIntPrV()
snap.GetSccSzCnt(graph, ComponentDist)
print(ComponentDist.Len())
```

```
68
CPU times: user 40.1 s, sys: 11 ms, total: 40.1 s
Wall time: 40.1 s
```

最大强连通网络的节点数和边数

```
%%time
#最大的强连通图的节点数和边数
MxWcc = snap.GetMxScc(graph)
print(MxWcc.GetNodes())
print(MxWcc.GetEdges())
```

```
3828682
65825429
CPU times: user 2min 39s, sys: 185 ms, total: 2min 39s
Wall time: 2min 40s
```

最大强连通网络的节点分数：

```
%%time
#最大的强连通图的节点相对大小
print(snap.GetMxSccSz(graph))
```

```
0.789814527729
CPU times: user 36.9 s, sys: 92 ms, total: 37 s
Wall time: 37 s
```

最大双连通图网络的节点数和边数：

```
%%time
#有向图的最大双连通图网络的节点数和边数
BiCon = snap.GetMxBiCon(graph)
print(BiCon.GetNodes())
print(BiCon.GetEdges())
```

```
3665291
67046693
CPU times: user 2min 33s, sys: 948 ms, total: 2min 34s
Wall time: 2min 34s
```

4.1.4. 互连三元祖和聚集方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/triads.html

互连三元祖闭合环个数：

```
%%time
#互连三元祖个数
print(snap.GetTriads(graph))
```

```
285730264
CPU times: user 11min 31s, sys: 671 ms, total: 11min 32s
Wall time: 11min 32s
```

互连三元祖闭合环个数和非闭合环个数

```
%%time
#互连三元祖闭合环个数和非闭合环个数
result = snap.GetTriadsAll(graph)
print "closed triads", result[0]
print "open triads", result[2]
```

```
closed triads 285730264
open triads 6412312961
CPU times: user 5min 48s, sys: 326 ms, total: 5min 48s
Wall time: 5min 48s
```

参与互连三元祖闭合环的边数：

```
%%time
#参与互连三元祖闭合环的边数
print(snap.GetTriadEdges(graph))
```

```
60549807
CPU times: user 3min 36s, sys: 0 ns, total: 3min 36s
Wall time: 3min 36s
```

按度分组聚集系数集合，总平均聚集系数、闭合环个数、非闭合环个数

```
%%time
#聚集系数
DegToCCfV = snap.TFltPrV()
result = snap.GetClustCfAll(graph, DegToCCfV)
print "Aggregation coefficient set grouped by degree",DegToCCfV.Len()
print "average clustering coefficient", result[0]
print "closed triads", result[1]
print "open triads", result[2]
```

```
Aggregation coefficient set grouped by degree 2045
average clustering coefficient 0.274241387656
closed triads 285730264
open triads 6412312961
CPU times: user 6min 2s, sys: 338 ms, total: 6min 2s
Wall time: 6min 2s
```

4.1.5. 图的广度和深度遍历方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/bfsdfs.html

图的近似直径值(也可以得到其对应的无向图的相应值)

```
%%time
#利用广度遍历来求得图的近似直径值
#将其看成是无向图
print(snap.GetBfsFullDiam(graph, 10, False))
#将其看成是有向图
print(snap.GetBfsFullDiam(graph, 10, True))
```

```
16
16
CPU times: user 8min 25s, sys: 218 ms, total: 8min 26s
Wall time: 8min 26s
```

90-th percentile of the distribution 的近似直径值(也可以得到其对应的无向图的相应值)

```
%%time
#th percentile of the distribution的近似直径值
#将其看成是无向图
print(snap.GetBfsEffDiam(graph, 10, False))
#将其看成是有向图
print(snap.GetBfsEffDiam(graph, 10, True))
```

```
6.26989395007
6.77949979352
CPU times: user 6min 45s, sys: 126 ms, total: 6min 45s
Wall time: 6min 45s
```

距离节点 1 两跳可以到达的节点集合(可以得到其对应的无向图的相应值)

```
%%time
#距离节点1两跳可以到达的节点集合
#将其看成是无向图
NodeVec = snap.TIntV()
snap.GetNodesAtHop(graph, 1, 2, NodeVec, False)
print(NodeVec.Len())
#将其看成是有向图
NodeVec = snap.TIntV()
snap.GetNodesAtHop(graph, 1, 2, NodeVec, True)
print(NodeVec.Len())
```

```
49735
16791
CPU times: user 180 ms, sys: 498 ms, total: 678 ms
Wall time: 665 ms
```

节点 1 和 2 的最短路径长度

```
%%time
#节点1和2的最短路径长度
#将其看成是无向图
print(snap.GetShortPath(graph,1,2,False))
#将其看成是有向图
print(snap.GetShortPath(graph,1,2,True))
```

```
2
2
CPU times: user 187 ms, sys: 159 ms, total: 346 ms
Wall time: 342 ms
```

最短路径分布的 90-th percentile

```
%%time
#最短路径分布的90-th percentile
print(snap.GetAnfEffDiam(graph))
```

```
6.28662930888
CPU times: user 8min 23s, sys: 805 ms, total: 8min 24s
Wall time: 8min 24s
```

以节点 1 为根节点得到到其他节点的树结构

```
%%time
#以节点1为根节点得到到其他节点的树结构
BfsTree = snap.GetBfsTree(graph, 1, True, False)
# for EI in BfsTree.Edges():
#     print "Edge from %d to %d in generated tree." % (EI.GetSrcNId(), EI.GetDstNId())
print(BfsTree.GetEdges())
```

```
14415513
CPU times: user 1min 53s, sys: 705 ms, total: 1min 54s
Wall time: 1min 54s
```

4.1.6. 节点中心性方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/centr.html

基于 NodeFrac 节点的样本计算节点和边的中心性近似值集合

```
%%time
#基于NodeFrac节点的样本计算节点和边的中心性近似值集合
Nodes = snap.TIntFltH()
Edges = snap.TIntPrFltH()
snap.GetBetweennessCentr(graph, Nodes, Edges, 1.0)
# for node in Nodes:
#     print "node: %d centrality: %f" % (node, Nodes[node])
# for edge in Edges:
#     print "edge: (%d, %d) centrality: %f" % (edge.GetVal1(), edge.GetVal2(), Edges[edge])
print(Nodes.Len())
print(Edges.Len())
```

时间过长，1 个多小时还没有出结果，强行停止了

节点 0 的 farness centrality：与给定节点位于同一连接组件中的所有其他节点的平均最短路径长度。

```
%%time
#节点0的farness centrality
print(snap.GetFarnessCentr(graph, 0))
```

```
4.27710142534
CPU times: user 7.49 s, sys: 49 ms, total: 7.54 s
Wall time: 7.54 s
```

节点 0 的 closeness centrality：是 farness centrality 的倒数

```
%%time
#节点0的closeness centrality: 是farness centrality的倒数
print(snap.GetClosenessCentr(graph, 0))
```

```
0.233803200007
CPU times: user 7.5 s, sys: 39 ms, total: 7.54 s
Wall time: 7.54 s
```

计算每个节点的 PageRank 分数集合

```
%%time
#计算每个节点的PageRank分数集合
PRankH = snap.TIntFltH()
snap.GetPageRank(graph, PRankH)
# for item in PRankH:
#     print item, PRankH[item]
print(PRankH.Len())
```

```
4847571
CPU times: user 53.9 s, sys: 113 ms, total: 54.1 s
Wall time: 54.1 s
```

计算每个节点的 Hubs 和 Authorities 分数集合

```
%%time
#计算每个节点的Hubs和Authorities分数集合
NIdHubH = snap.TIntFltH()
NIdAuthH = snap.TIntFltH()
snap.GetHits(graph, NIdHubH, NIdAuthH)
# for item in NIdHubH:
#     print item, NIdHubH[item]
# for item in NIdAuthH:
#     print item, NIdAuthH[item]
print(NIdHubH.Len())
print(NIdAuthH.Len())
```

```
4847571
4847571
CPU times: user 11min 31s, sys: 934 ms, total: 11min 32s
Wall time: 11min 32s
```

id 为 0 这个节点的 node eccentricity:节点偏心率，即从节点 0 到图中任何其他节点的最大最短路径距离

```
%%time
#id为0这个节点的node eccentricity:节点偏心率，即从节点0到图中任何其他节点的最大最短路径距离
#将其看成是无向图
print(snap.GetNodeEcc(graph,0, False))
#将其看成是有向图
print(snap.GetNodeEcc(graph,0, True))
```

```
11
14
CPU times: user 38.5 s, sys: 82 ms, total: 38.6 s
Wall time: 38.6 s
```

4.1.7. 邻接矩阵方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/svd.html

有向图的邻接矩阵的 SngVals 最大奇异值

```
%%time
#计算有向图的邻接矩阵的SngVals最大奇异值。
SngVals = 3
SngValV = snap.TFltV()
snap.GetSngVals(graph, SngVals, SngValV)
for item in SngValV:
    print item
```

```
405.583770644
359.899654624
332.924916517
CPU times: user 10min 33s, sys: 911 ms, total: 10min 34s
Wall time: 10min 34s
```

有向图的邻接矩阵的左右奇异向量

```
%%time
#计算有向图的邻接矩阵的左右奇异向量
LeftSV = snap.TFltV()
RightSV = snap.TFltV()
snap.GetSngVec(graph, LeftSV, RightSV)
# print "Left singular vector:"
# for item in LeftSV:
#     print item
# print "Right singular vector:"
# for item in RightSV:
#     print item
```

```
CPU times: user 3min 48s, sys: 306 ms, total: 3min 49s
Wall time: 3min 49s
```

4.1.8. K-core 方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/kcore.html

计算 K-score 为 K 的网络集合

```
%%time
#计算K-score为K的网络集合
K = 100
KCore = snap.GetKCore(graph, K)
if KCore.Empty():
    print 'No Core exists for K=%d' % K
else:
    print 'Core exists for K=%d' % K
```

```
Core exists for K=100
CPU times: user 36.4 s, sys: 249 ms, total: 36.7 s
Wall time: 36.7 s
```

计算得到每个 K-score 中的节点数

```
%%time
#计算得到每个K-score中的边数
CoreIDSzV = snap.TIntPrV()
kValue = snap.GetKCoreEdges(Ugraph, CoreIDSzV)
# for item in CoreIDSzV:
#     print "order: %d edges: %d" % (item.GetVal1(), item.GetVal2())
```

```
CPU times: user 47 s, sys: 163 ms, total: 47.1 s
Wall time: 47.1 s
```

```
%%time
#计算得到每个K-score中的节点数
CoreIDSzV = snap.TIntPrV()
kValue = snap.GetKCoreNodes(graph, CoreIDSzV)
# for item in CoreIDSzV:
#     print "order: %d nodes: %d" % (item.GetVal1(), item.GetVal2())
```

```
CPU times: user 2min 13s, sys: 1.2 s, total: 2min 14s
Wall time: 2min 14s
```

计算得到每个 K-score 中的边数

4.2. 无向数据集

Friendster (http://snap.stanford.edu/data/com-Friendster.html)
一共包含三个数据集：

File	Description
com-friendster.ungraph.txt.gz	Undirected Friendster network
com-friendster.all.cmty.txt.gz	Friendster communities
com-friendster.top5000.cmty.txt.gz	Friendster communities (Top 5,000)

第一个文件形式是：(源节点->目的节点)，大约 31G

```
# Undirected graph: ../../data/output/friendster.txt
# Friendster
# Nodes: 65608366 Edges: 1806067135
# FromNodeId    ToNodeId
101     102
101     104
101     107
101     125
101     165
101     168
101     170
101     176
101     180
101     181
101     182
```

后面两个文件形式：[source node id,destination node name 1,destination node name 2,.........]

```
new 1   com-friendster.top5000.cmty.txt
4025014 4091805 4238146 4765046 5039862 5071740 5454285 5729655 6656360 7064157 7284003 7674574 7732563 8001495 8020524 8442118 8455737 8561300 85
58993749    58993916    58993965    58994004
58993749    58993916    58993965    58994004
5291072 6456009 7904202 17937941    18101539    18302673    19548152    20253288    20579061    20820778    20887408
19511   59580   153031  1993538 2285646 3017978 3102372 19324643
42096585    42512480    45564113    46091893
1505575 2075208 2464772 4881814 5241619 17891430
1561757 2064256 6551656 11904965    14296652    17607975    17623618
1840109 3949434 5552823 16920962    17278379    18198408
1174735 1175824 2903686 4703797 18232338
284628  670678  698374  3731862 8035663 16450484    16585822    17351361    17389661    17533723    17644470    17697362    17812797    18173791
51863864    114321645   114322493   114322660
```

故加载时对应的 API 是：

LoadConnList 和 LoadConnListStr

由于第一个数据集太大，故这里实践的是第二个数据集

加载数据集

```
%%time
#加载数据: http://snap.stanford.edu/data/com-Friendster.html
Ugraph = snap.LoadConnList(snap.PUNGraph, "com-friendster.all.cmty.txt")
```

```
CPU times: user 2min 18s, sys: 1.46 s, total: 2min 20s
Wall time: 2min 20s
```

4.2.1. 图的基本常见的统计信息

可以使用 print 函数，其会自动计算一些基本的统计信息

```
%%time
snap.PrintInfo(Ugraph,"Python type PUNGraph","info-pungraph.txt")
```

```
CPU times: user 1.07 s, sys: 3 ms, total: 1.07 s
Wall time: 1.1 s
```

---

jupyter info-pungraph.txt✔ 几秒前

File Edit View Language

```
Python type PUNGraph:
  Nodes:                    7944949
  Edges:                    20783768
  Zero Deg Nodes:           0
  Zero InDeg Nodes:         0
  Zero OutDeg Nodes:        0
  NonZero In-Out Deg Nodes: 7944949
```

======================以下是更多指标详细的统计过程===========================

4.2.2. 度节点方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/degree.html

节点数

```
%%time
#图中节点个数
print(Ugraph.GetNodes())
```

```
7944949
CPU times: user 999 µs, sys: 999 µs, total: 2 ms
Wall time: 1.52 ms
```

边数

```
%%time
#图中边数
print(Ugraph.GetEdges())
```

```
20783768
CPU times: user 999 µs, sys: 2 ms, total: 3 ms
Wall time: 1.88 ms
```

度为 100 的节点个数

```
%%time
#度为100的节点个数
print(snap.CntDegNodes(Ugraph, 100))
```

```
410
CPU times: user 1.32 s, sys: 4 ms, total: 1.33 s
Wall time: 1.43 s
```

出度为 100 的节点个数

入度为 100 的节点个数

对于无向图来说没有所谓的入度和出度，调用相应的 API 时，其会将边全部看做入度或者出度

```
%%time
#出度为100的节点个数
print(snap.CntOutDegNodes(Ugraph, 100))
```

```
410
CPU times: user 1.08 s, sys: 1e+03 µs, total: 1.08 s
Wall time: 1.08 s
```

```
%%time
#入度为100的节点个数
print(snap.CntInDegNodes(Ugraph, 100))
```

```
410
CPU times: user 971 ms, sys: 1 ms, total: 972 ms
Wall time: 969 ms
```

有着最大出度的节点的 id

有着最大入度的节点的 id

同理调用上述 API 时其也会分别将边全部视为入度边或出度边

```
%%time
#有着最大出度的节点的id
print(snap.GetMxOutDegNId(Ugraph))
```

```
3922991
CPU times: user 1.34 s, sys: 2 ms, total: 1.34 s
Wall time: 1.36 s
```

```
%%time
#有着最大入度的节点的id
print(snap.GetMxInDegNId(Ugraph))
```

```
3922991
CPU times: user 947 ms, sys: 6 ms, total: 953 ms
Wall time: 952 ms
```

4.2.3. 图连通方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/cncom.html

对于无向图，强连通和弱连通是一样的

弱连通网络的个数：

强连通网络的个数：

```
%%time
#Components是弱连通网络集合
Components = snap.TCnComV()
snap.GetWccs(Ugraph, Components)
print(Components.Len())
```

```
42236
CPU times: user 43.1 s, sys: 253 ms, total: 43.3 s
Wall time: 43.4 s
```

```
%%time
#Components是强连通网络集合
Components = snap.TCnComV()
snap.GetSccs(Ugraph, Components)
print(Components.Len())
```

```
42236
CPU times: user 1min 1s, sys: 913 ms, total: 1min 2s
Wall time: 1min 2s
```

弱连通网络的种数
强连通网络的种数

```
%%time
#ComponentDist相当于对Components按(弱连通网络中节点数大小)进行了分组
ComponentDist = snap.TIntPrV()
snap.GetWccSzCnt(Ugraph, ComponentDist)
print(ComponentDist.Len())
```

```
37
CPU times: user 35 s, sys: 273 ms, total: 35.3 s
Wall time: 35.3 s
```

```
%%time
#ComponentDist相当于对Components按(强连通网络中节点数大小)进行了分组
ComponentDist = snap.TIntPrV()
snap.GetSccSzCnt(Ugraph, ComponentDist)
print(ComponentDist.Len())
```

```
37
CPU times: user 55.4 s, sys: 279 ms, total: 55.6 s
Wall time: 55.7 s
```

最大弱连通网络的节点数和边数
最大强连通网络的节点数和边数

```
%%time
#最大的弱连通图节点数和边数
MxWcc = snap.GetMxWcc(Ugraph)
print(MxWcc.GetNodes())
print(MxWcc.GetEdges())
```

```
7835772
20716047
CPU times: user 2min 9s, sys: 1.15 s, total: 2min 10s
Wall time: 2min 10s
```

```
%%time
#最大的强连通图的节点数和边数
MxWcc = snap.GetMxScc(Ugraph)
print(MxWcc.GetNodes())
print(MxWcc.GetEdges())
```

```
7835772
20716047
CPU times: user 4min 17s, sys: 1.55 s, total: 4min 19s
Wall time: 4min 19s
```

最大弱连通网络的节点分数
最大强连通网络的节点分数

```
%%time
#最大的强连通图的节点相对大小
print(snap.GetMxSccSz(Ugraph))
```

```
0.986258313301
CPU times: user 1min 17s, sys: 18 ms, total: 1min 17s
Wall time: 1min 17s
```

```
%%time
#最大的强连通图的节点相对大小
print(snap.GetMxSccSz(Ugraph))
```

```
0.986258313301
CPU times: user 1min 17s, sys: 18 ms, total: 1min 17s
Wall time: 1min 17s
```

和节点 4025014 在同一个弱连通网路的节点集合

```
%%time
#和节点4025014在同一个弱连通网路的节点集合
CnCom = snap.TIntV()
snap.GetNodeWcc(Ugraph, 4025014, CnCom)
# print "Nodes in the same connected component as node 0:"
# for node in CnCom:
#     print node
# print('--')
print(CnCom.Len())
```

```
7835772
CPU times: user 24.4 s, sys: 1e+03 µs, total: 24.4 s
Wall time: 24.4 s
```

最大双连通图网络的节点数和边数

```
%%time
#最大的弱连通图节点数和边数
MxWcc = snap.GetMxWcc(Ugraph)
print(MxWcc.GetNodes())
print(MxWcc.GetEdges())
```

```
7835772
20716047
CPU times: user 2min 9s, sys: 1.15 s, total: 2min 10s
Wall time: 2min 10s
```

双连通和强连通是一样的，只不过前者多用于无向图，后者多使用于有向图

articulation points:返回无向图的连接点集合,输入只能是无向图，解释如下：

https://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/

```
%%time
#articulation points:返回无向图的连接点集合
ArtNIdV = snap.TIntV()
snap.GetArtPoints(Ugraph, ArtNIdV)
print(ArtNIdV.Len())
```

```
615584
CPU times: user 1min 6s, sys: 41 ms, total: 1min 6s
Wall time: 1min 6s
```

edge bridges:类似于 articulation points，只不过是针对边说的，输入只能是无向图

```
%%time
#edge bridges:类似articulation points, 只不过是针对边说的
EdgeV = snap.TIntPrV()
snap.GetEdgeBridges(Ugraph, EdgeV)
print(EdgeV.Len())
```

```
3534315
CPU times: user 4min 33s, sys: 465 ms, total: 4min 33s
Wall time: 4min 33s
```

4.2.4. 互连三元祖和聚集方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/triads.html

互连三元祖闭合环个数：

```
%%time
#互连三元祖闭合环个数
print(snap.GetTriads(Ugraph))
```

```
6169038
CPU times: user 7min 47s, sys: 853 ms, total: 7min 48s
Wall time: 7min 48s
```

互连三元祖闭合环个数和非闭合环个数

```
%%time
#互连三元祖闭合环个数和非闭合环个数
result = snap.GetTriadsAll(Ugraph)
print "closed triads", result[0]
print "open triads", result[2]
```

```
closed triads 6169038
open triads 10439748758
CPU times: user 8min 37s, sys: 641 ms, total: 8min 38s
Wall time: 8min 38s
```

参与互连三元祖闭合环的边数：

```
%%time
#参与互连三元祖闭合环的边数
print(snap.GetTriadEdges(Ugraph))
```

```
7243985
CPU times: user 18min 50s, sys: 7 ms, total: 18min 50s
Wall time: 18min 51s
```

按度分组聚集系数集合，总平均聚集系数、闭合环个数、非闭合环个数

```
%%time
#按度分组聚集系数集合，总平均聚集系数、闭合环个数、非闭合环个数
DegToCCfV = snap.TFltPrV()
result = snap.GetClustCfAll(Ugraph, DegToCCfV)
print "Aggregation coefficient set grouped by degree",DegToCCfV.Len()
print "average clustering coefficient", result[0]
print "closed triads", result[1]
print "open triads", result[2]
```

```
Aggregation coefficient set grouped by degree 2100
average clustering coefficient 0.0867827397672
closed triads 6169038
open triads 10439748758
CPU times: user 6min 41s, sys: 874 ms, total: 6min 42s
Wall time: 6min 42s
```

4.2.5. 图的广度和深度遍历方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/bfsdfs.html

图的近似直径值(也可以得到其对应的无向图的相应值)

```
%%time
#利用广度遍历来求得图的近似直径值
print(snap.GetBfsFullDiam(Ugraph, 10, False))
```

```
19
CPU times: user 7min 35s, sys: 153 ms, total: 7min 35s
Wall time: 7min 35s
```

91-th percentile of the distribution 的近似直径值(也可以得到其对应的无向图的相应值)

```
%%time
#距离节点4025014两跳可以到达的节点集合
NodeVec = snap.TIntV()
snap.GetNodesAtHop(Ugraph, 4025014, 2, NodeVec, False)
print(NodeVec.Len())
```

```
4132
CPU times: user 497 ms, sys: 1 ms, total: 498 ms
Wall time: 496 ms
```

距离节点 4025014 两跳可以到达的节点集合(可以得到其对应的无向图的相应值)


```
%%time
#距离节点4025014两跳可以到达的节点集合
NodeVec = snap.TIntV()
snap.GetNodesAtHop(Ugraph, 4025014, 2, NodeVec, False)
print(NodeVec.Len())
```

```
4132
CPU times: user 497 ms, sys: 1 ms, total: 498 ms
Wall time: 496 ms
```


节点 4025014 和 114322660 的最短路径长度

```
%%time
#节点4025014和114322660的最短路径长度
print(snap.GetShortPath(Ugraph, 4025014, 114322660, False))
```

```
4
CPU times: user 3.48 s, sys: 0 ns, total: 3.48 s
Wall time: 3.48 s
```

最短路径分布的 90-th percentile

```
%%time
#最短路径分布的90-th percentile
print(snap.GetAnfEffDiam(Ugraph))
```

```
6.49528180031
CPU times: user 6min 51s, sys: 425 ms, total: 6min 52s
Wall time: 6min 52s
```

以节点 1 为根节点得到到其他节点的树结构

```
%%time
#以节点4025014为根节点得到到其他节点的树结构
BfsTree = snap.GetBfsTree(Ugraph, 4025014, True, False)
# for EI in BfsTree.Edges():
#     print "Edge from %d to %d in generated tree." % (EI.GetSrcNId(), EI.GetDstNId())
print(BfsTree.GetEdges())
```

```
13658750
CPU times: user 1min 29s, sys: 559 ms, total: 1min 30s
Wall time: 1min 30s
```

4.2.6. 节点中心性方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/centr.html

基于 NodeFrac 节点的样本计算节点和边的中心性近似值集合

时间过长，1 个半小时还没有出结果，强行停止了

节点 4025014 的 farness centrality：与给定节点位于同一连接组件中的所有其他节点的平均最短路径长度。

```
%%time
#节点4025014的farness centrality
print(snap.GetFarnessCentr(Ugraph, 4025014))
```

```
4.68962640778
CPU times: user 10 s, sys: 85 ms, total: 10.1 s
Wall time: 10.1 s
```

节点 4025014 的 closeness centrality：是 farness centrality 的倒数

```
%%time
#节点4025014的closeness centrality: 是farness centrality的倒数
print(snap.GetClosenessCentr(Ugraph, 4025014))
```

```
0.213236602033
CPU times: user 10.1 s, sys: 154 ms, total: 10.2 s
Wall time: 10.3 s
```

计算每个节点的 PageRank 分数集合

```
%%time
#计算每个节点的PageRank分数集合
PRankH = snap.TIntFltH()
snap.GetPageRank(Ugraph, PRankH)
# for item in PRankH:
#     print item, PRankH[item]
print(PRankH.Len())
```

```
7944949
CPU times: user 1min 52s, sys: 545 ms, total: 1min 53s
Wall time: 1min 53s
```

计算每个节点的 Hubs 和 Authorities 分数集合

```
%%time
#计算每个节点的Hubs和Authorities分数集合
NIdHubH = snap.TIntFltH()
NIdAuthH = snap.TIntFltH()
snap.GetHits(Ugraph, NIdHubH, NIdAuthH)
# for item in NIdHubH:
#     print item, NIdHubH[item]
# for item in NIdAuthH:
#     print item, NIdAuthH[item]
print(NIdHubH.Len())
print(NIdAuthH.Len())
```

```
7944949
7944949
CPU times: user 19min 30s, sys: 704 ms, total: 19min 31s
Wall time: 19min 31s
```

id 为 4025014 这个节点的 node eccentricity:节点偏心率，即从节点 0 到图中任何其他节点的最大最短路径距离

```
%%time
#id为4025014这个节点的node eccentricity:节点偏心率, 即从节点0到图中任何其他节点的最大最短路径距离
print(snap.GetNodeEcc(Ugraph, 4025014, False))
```

```
16
CPU times: user 24.3 s, sys: 84 ms, total: 24.4 s
Wall time: 24.4 s
```

4.2.7. 社区发现方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/community.html
关于社区方面概念的介绍：https://blog.csdn.net/cleverlzc/article/details/39494957
该部分 1 个小时还没有出结果，停止运行了

社区的集合(Clauset-Newman-Moore community detection method 算法)

```
%%time
#社区的集合: 关于社区的概念https://blog.csdn.net/cleverlzc/article/details/39494957
#使用的是Clauset-Newman-Moore community detection method算法
CmtyV = snap.TCnComV()
modularity = snap.CommunityCNM(Ugraph, CmtyV)
# for Cmty in CmtyV:
#     print "Community: "
#     for NI in Cmty:
#         print NI
print "The modularity of the network is %f" % modularity
print(CmtyV.Len())
```

社区的集合(Girvan-Newman community detection algorithm 算法)

```
#社区的集合
#使用的是 Girvan-Newman community detection algorithm算法
CmtyV = snap.TCnComV()
modularity = snap.CommunityGirvanNewman(Ugraph, CmtyV)
# for Cmty in CmtyV:
#     print "Community: "
#     for NI in Cmty:
#         print NI
print "The modularity of the network is %f" % modularity
print(CmtyV.Len())
```

4.2.8. 邻接矩阵方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/svd.html

无向图邻接矩阵前 EigVals 个的特征值集合

```
:  %%time
   #计算无向图邻接矩阵前EigVals个的特征值集合
   EigVals = 3
   PEigV = snap.TFltV()
   snap.GetEigVals(Ugraph, EigVals, PEigV)
   for item in PEigV:
       print item

   387.264224046
   256.192033973
   -381.681435039
   CPU times: user 3min 17s, sys: 1.18 s, total: 3min 18s
   Wall time: 3min 18s
```

无向图邻接矩阵的第一列特征向量集合

```
]: %%time
   #计算计算无向图邻接矩阵的第一列特征向量集合
   EigVec =  snap.TFltV()
   snap.GetEigVec(Ugraph, EigVec)
   # for Val in EigVec:
   #     print Val
   print(EigVec.Len())

    7944949
   CPU times: user 4min 29s, sys: 1.74 s, total: 4min 31s
   Wall time: 4min 31s
```

反向参与率

```
:  %%time
   #计算Inverse participation ratio, 最大搜索时间50秒, 最大的返回特征数量数为5
   EigValIprV = snap.TFltPrV()
   snap.GetInvParticipRat(Ugraph, 5, 50, EigValIprV)
   for item in EigValIprV:
       print '%f, %f' % (item.GetVal1(), item.GetVal2())

   -363.123467, 0.191928
   -28.712595, 0.002063
   50.858960, 0.004579
   370.633989, 0.174107
   CPU times: user 4min 4s, sys: 3.27 s, total: 4min 8s
   Wall time: 4min 8s
```

4.2.9. K-core 方面的属性

该部分 API:http://snap.stanford.edu/snappy/doc/reference/kcore.html

计算 K-score 为 K 的网络集合

```
%%time
#计算K-score为K的网络集合
K = 100
KCore = snap.GetKCore(Ugraph, K)
if KCore.Empty():
    print 'No Core exists for K=%d' % K
else:
    print 'Core exists for K=%d' % K
```

```
No Core exists for K=100
CPU times: user 17.7 s, sys: 85 ms, total: 17.8 s
Wall time: 17.8 s
```

计算得到每个 K-score 中的节点数

```
%%time
#计算得到每个K-score中的节点数
CoreIDSzV = snap.TIntPrV()
kValue = snap.GetKCoreNodes(Ugraph, CoreIDSzV)
# for item in CoreIDSzV:
#     print "order: %d nodes: %d" % (item.GetVal1(), item.GetVal2())
```

```
CPU times: user 32.9 s, sys: 170 ms, total: 33.1 s
Wall time: 33.1 s
```

计算得到每个 K-score 中的边数

```
%%time
#计算得到每个K-score中的边数
CoreIDSzV = snap.TIntPrV()
kValue = snap.GetKCoreEdges(Ugraph, CoreIDSzV)
# for item in CoreIDSzV:
#     print "order: %d edges: %d" % (item.GetVal1(), item.GetVal2())
```

```
CPU times: user 47 s, sys: 163 ms, total: 47.1 s
Wall time: 47.1 s
```

****

****