```python
G = snap.LoadEdgeList(snap.PNGraph, "qa.txt", 1, 5)
Snap.PrintInfo(G, "QA Stats", "qa-info.txt", False)
```

**Output:**

<table>
<thead>
<tr>
<th>QA Stats: Directed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes:</td>
</tr>
<tr>
<td>Edges:</td>
</tr>
<tr>
<td>Zero Deg Nodes:</td>
</tr>
<tr>
<td>Zero InDeg Nodes:</td>
</tr>
<tr>
<td>Zero OutDeg Nodes:</td>
</tr>
<tr>
<td>NonZero In-Out Deg Nodes:</td>
</tr>
<tr>
<td>Unique directed edges:</td>
</tr>
<tr>
<td>Unique undirected edges:</td>
</tr>
<tr>
<td>Self Edges:</td>
</tr>
<tr>
<td>BiDir Edges:</td>
</tr>
<tr>
<td>Closed triangles:</td>
</tr>
<tr>
<td>Open triangles:</td>
</tr>
<tr>
<td>Frac. of closed triads:</td>
</tr>
<tr>
<td>Connected component size:</td>
</tr>
<tr>
<td>Strong conn. comp. size:</td>
</tr>
<tr>
<td>Approx. full diameter:</td>
</tr>
<tr>
<td>90% effective diameter:</td>
</tr>
</tbody>
</table>
Basic Graph Generators

- Complete, circle, grid, star, tree graphs

\[ GG = \text{snap.GenGrid} (\text{snap.PUNGraph}, 4, 3) \]
\[ GT = \text{snap.GenTree} (\text{snap.PUNGraph}, 4, 2) \]
Advanced Graph Generators

- Erdos-Renyi, Preferential attachment,
- Forest Fire, Small-world, Configuration model
- Kronecker, RMat, Graph rewiring

\[
\text{GPA} = \text{snap.GenPrefAttach}(30, 3, \text{snap.TRnd}())
\]
Subgraphs and Conversions

- Extract subgraphs, convert from one graph type to another

Get an induced subgraph on a set of nodes $NIdV$:

$$NIdV = \text{snap.TIntV}()$$
for $i$ in range(1, 9): $NIdV$.Add($i$)

$$\text{SubGPA} = \text{snap.GetSubGraph}(\text{GPA, NIdV})$$
Connected Components

- **Analyze graph connectedness**
  - Strongly and Weakly connected components
    - Test connectivity, get sizes, get components, get largest
    - Articulation points, bridges
  - Bi-connected, 1-connected

```python
MxWcc = snap.GetMxWcc(G)  # Get largest WCC
print "max wcc nodes %d, edges %d" %
    (MxWcc.GetNodes(), MxWcc.GetEdges())

WccV = snap.TIntPrV()  # Get WCC sizes
snap.GetWccSzCnt(G, WccV)

print "# of connected component sizes", WccV.Len()
for comp in WccV:
    print "size %d, number of components %d" %
        (comp.GetVal1(), comp.GetVal2())
```
Node Degrees

- **Analyze node connectivity**
  - Find node degrees, maximum degree, degree distribution
  - In-degree, out-degree, combined degree

```python
NId = snap.GetMxDegNId(GPA)
print "max degree node", NId

DegToCntV = snap.TIntPrV()
snap.GetDegCnt(GPA, DegToCntV)
for item in DegToCntV:
    print "%d nodes with degree %d" % (item.GetVal2(), item.GetVal1())
```

max degree node 1
13 nodes with degree 3
4 nodes with degree 4
3 nodes with degree 5
2 nodes with degree 6
1 nodes with degree 7
1 nodes with degree 9
2 nodes with degree 10
2 nodes with degree 11
1 nodes with degree 13
1 nodes with degree 15
Node Centrality

- **Find “importance” of nodes in a graph**
  - PageRank, Hubs and Authorities
  - Degree-, betweenness-, closeness-, farness-, and eigen-centrality

```python
PRankH = snap.TIntFltH()
snap.GetPageRank(G, PRankH)

for item in PRankH:
    print item, PRankH[item]
```

- Calculate node PageRank scores
- Print them out
Triads and Clustering Coefficient

- Analyze connectivity among the neighbors
  - # of triads, fraction of closed triads
  - Fraction of connected neighbor pairs
  - Graph-based, node-based

Triads = snap.GetTriads(GPA)
print "triads", Triads

CC = snap.GetClustCf(GPA)
print "clustering coefficient", CC
Breadth and Depth First Search

- **Distances between nodes**
  - Diameter, Effective diameter
  - Shortest path, Neighbors at distance \( d \)
  - Approximate neighborhood (not BFS based)

```python
D = snap.GetBfsFullDiam(G, 100)
print "diameter", D

ED = snap.GetBfsEffDiam(G, 100)
print "effective diameter", ED
```

Calculate diameter

Calculate effective diameter
Community Detection

- Identify communities of nodes
  - Clauset-Newman-Moore, Girvan-Newman
    - Can be compute time intensive
  - BigClam, CODA, Cesna (C++ only)

```python
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
```

Clauset-Newman-Moore
Spectral properties of a graph

- Calculations based on graph adjacency matrix
  - Get Eigenvalues, Eigenvectors
  - Get Singular values, leading singular vectors

```python
EigV = snap.TFltV()
snap.GetEigVec(G, EigV)
	nr = 0

for f in EigV:
    nr += 1
    print "%d: %.6f" % (nr, f)
```

Get leading eigenvector
K-core decomposition

- Repeatedly remove nodes with low degrees
  - Calculate K-core

\[ \text{Core3} = \text{snap.GetKCore}(G, 3) \]

Calculate 3-core