An Introduction to Snap.py: SNAP for Python

http://snap.stanford.edu/snappy

CS224W, Fall 2019

Snap.py Tutorial: Content

- Introduction to SNAP
- Snap.py for Python
- Network analytics

What is SNAP?

- **S**tanford **N**etwork **A**nalysis **P**latform (SNAP) is a general purpose, high-performance system for analysis and manipulation of large networks
  - http://snap.stanford.edu
  - Scales to massive networks with hundreds of millions of nodes and billions of edges

- **SNAP software**
  - Snap.py for Python, SNAP C++
- **SNAP datasets**
  - Over 70 network datasets

Snap.py Resources

- **Prebuilt packages** available for Mac OS X, Windows, Linux
  http://snap.stanford.edu/snappy/index.html
- **Snap.py documentation**:
  http://snap.stanford.edu/snappy/doc/index.html
  - Quick Introduction, Tutorial, Reference Manual
- **SNAP user mailing list**
  http://groups.google.com/group/snap-discuss
- **Developer resources**
  - Software available as open source under BSD license
  - GitHub repository
    https://github.com/snap-stanford/snap-python

SNAP C++ Resources

- **Source code** available for Mac OS X, Windows, Linux
  http://snap.stanford.edu/snap/download.html
- **SNAP documentation**
  http://snap.stanford.edu/snap/doc.html
  - Quick Introduction, User Reference Manual
  - Source code, see **tutorials**
- **SNAP user mailing list**
  http://groups.google.com/group/snap-discuss
- **Developer resources**
  - Software available as open source under BSD license
  - GitHub repository
    https://github.com/snap-stanford/snap
  - SNAP C++ Programming Guide

SNAP Network Datasets

Collection of over 70 social network datasets:
http://snap.stanford.edu/data

Mailing list: http://groups.google.com/group/snap-datasets

- **Social networks:** online social networks, edges represent interactions between people
- **Twitter and Memetracker :** Memetracker phrases, links and 467 million Tweets
- **Citation networks:** nodes represent papers, edges represent citations
- **Collaboration networks:** nodes represent scientists, edges represent collaborations (co-authoring a paper)
- **Amazon networks :** nodes represent products and edges link commonly co-purchased products

What is Snap.py?

- **Snap.py** (pronounced "snappy"): **SNAP for Python**

  http://snap.stanford.edu/snappy

User Code	Python
Snap.py	Python
SNAP	C++

Solution	Fast Execution	Easy to use, interactive
C++	✓	
Python		✓
Snap.py (C++, Python)	✓	✓

Installing Snap.py

- **Installation:**
  - Follow instructions on the Snap.py webpage

```
pip install snap-stanford
```

If you encounter problems, please report them on Piazza

Installation Matrix

Operating System 64-bit	Python Environment	Python Version 64-bit	Install Method	Install Command
macOS 10.14	default system	Python 2.7	pip	sudo pip install snap-stanford
macOS 10.14	Homebrew	Python 3.7	pip	pip3 install snap-stanford
macOS 10.14	Anaconda	Python 3.7	conda	conda install -c snap-stanford snap-stanford
macOS 10.14	Miniconda	Python 3.7	pip	pip install http://snap.stanford.edu/snappy/release/snap_stanford-5.0.0-cp37-cp37m-macosx_10_7_x86_64.whl
macOS 10.13	default system	Python 2.7	setup.py	(download the package, unpack) sudo python setup.py install
macOS 10.13	Homebrew	Python 3.7	setup.py	(download the package, unpack) python3 setup.py install
Ubuntu 18.04	default system	Python 2.7	pip	sudo pip install snap-stanford
Ubuntu 18.04	apt-get	Python 3.[567]	pip	sudo pip3 install snap-stanford
Ubuntu 16.04	default system	Python 2.7	pip	sudo pip install snap-stanford
Ubuntu 16.04	apt-get	Python 3.[567]	pip	sudo pip3 install snap-stanford
Windows 10	python.org	Python 2.7	pip	pip install snap-stanford
Windows 10	python.org	Python 3.7	pip	pip install snap-stanford
CentOS 6.x	default system	Python 2.6	setup.py	(download the package, unpack) sudo python setup.py install

https://docs.google.com/spreadsheets/d/1m-5gHUmGzh8XfLUCAY3eYvdcBA98TUMMusVZkwmpdaI/edit?usp=sharing

Snap.py: Important

- The most important step for using Snap.py: **Import the snap module!**

```
$ python
>>> import snap
```

Snap.py Tutorial

- **On the Web:** http://snap.stanford.edu/snappy/doc/tutorial/index-tut.html
- **We will cover:**
  - Basic Snap.py data types
  - Vectors, hash tables and pairs
  - Graphs and networks
  - Graph creation
  - Adding and traversing nodes and edges
  - Saving and loading graphs
  - Plotting and visualization

Snap.py Naming Conventions (1)

Variable types/names:

- **...Int**: an integer operation, variable: **GetValInt()**
- **...Flt**: a floating point operation, variable; **GetValFlt()**
- **...Str**: a string operation, variable; **GetDateStr()**

Classes vs. Graph Objects:

- **T...**: a class type; **TUNGraph**
- **P...**: type of a graph object; **PUNGraph**

Data Structures:

- **...V:** a vector, variable **TIntV InNIdV**
- **...VV:** a vector of vectors (i.e., a matrix), variable **FltVV**

  TFltVV ... a matrix of floating point elements
- **...H**: a hash table, variable **NodeH**

  TIntStrH ... a hash table with **TInt** keys, **TStr** values
- **...HH**: a hash of hashes, variable **NodeHH**

  TIntIntHH ... a hash table with **TInt** key 1 and **TInt** key 2
- **...Pr:** a pair; type **TIntPr**

Snap.py Naming Conventions (2)

- **Get...**: an access method, **GetDeg()**
- **Set...**: a set method, **SetXYLabel()**
- **...I**: an iterator, **NodeI**
- **Id**: an identifier, **GetUId()**
- **NId**: a node identifier, **GetNId()**
- **EId**: an edge identifier, **GetEId()**
- **Nbr**: a neighbor, **GetNbrNId()**
- **Deg**: a node degree, **GetOutDeg()**
- **Src**: a source node, **GetSrcNId()**
- **Dst**: a destination node, **GetDstNId()**

Basic Types in Snap.py (and SNAP)

- **`TInt`**: Integer
- **`TFlt`**: Float
- **`TStr`**: String

- Used primarily for constructing composite types
- In general no need to deal with the basic types explicitly
  - Data types are automatically converted between C++ and Python
  - An illustration of explicit manipulation:

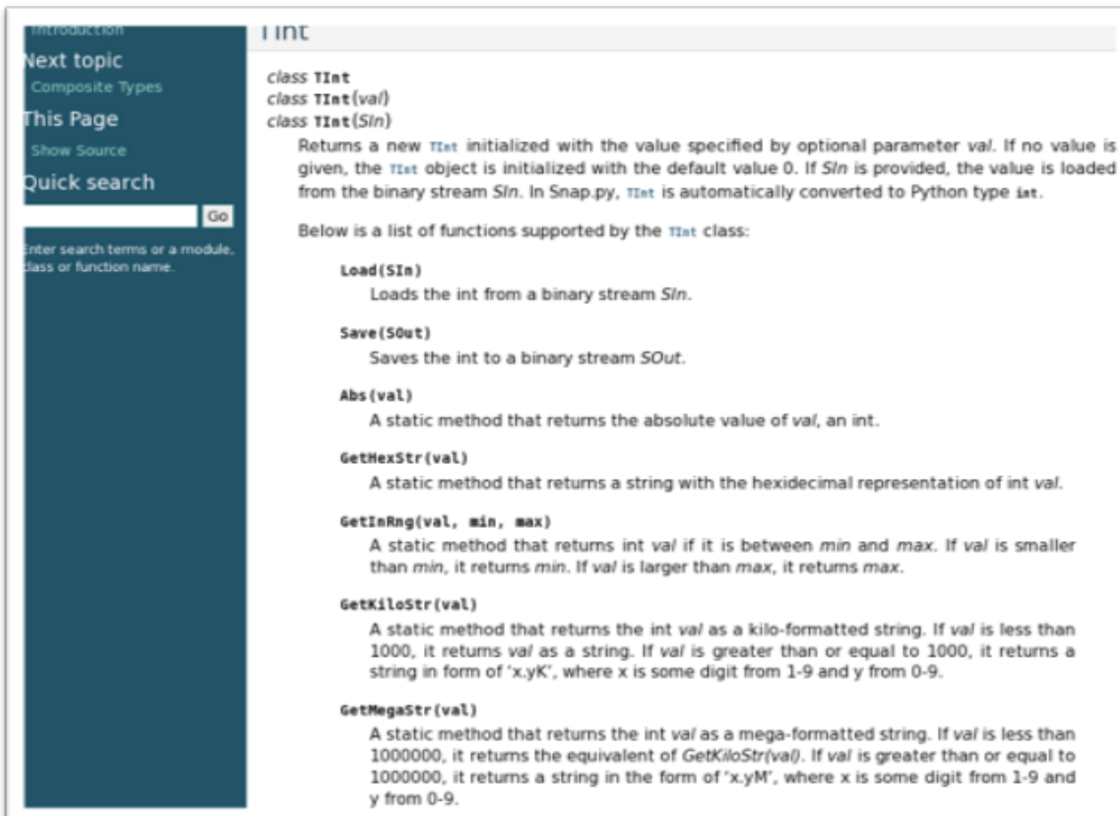
```
>>> i = snap.TInt(10)
>>> print i.Val
10
```

- **Note:** do not use an empty string “” in TStr parameters

Snap.py Reference Documentation

For more information check out Snap.py Reference Manual

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



Introduction

Next topic

Composite Types

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

TInt

class `TInt`

class `TInt`(*val*)

class `TInt`(*SIn*)

Returns a new `TInt` initialized with the value specified by optional parameter *val*. If no value is given, the `TInt` object is initialized with the default value 0. If *SIn* is provided, the value is loaded from the binary stream *SIn*. In Snap.py, `TInt` is automatically converted to Python type `int`.

Below is a list of functions supported by the `TInt` class:

`Load(SIn)`
Loads the int from a binary stream *SIn*.

`Save(SOut)`
Saves the int to a binary stream *SOut*.

`Abs(val)`
A static method that returns the absolute value of *val*, an int.

`GetHexStr(val)`
A static method that returns a string with the hexidecimal representation of int *val*.

`GetInRng(val, min, max)`
A static method that returns int *val* if it is between *min* and *max*. If *val* is smaller than *min*, it returns *min*. If *val* is larger than *max*, it returns *max*.

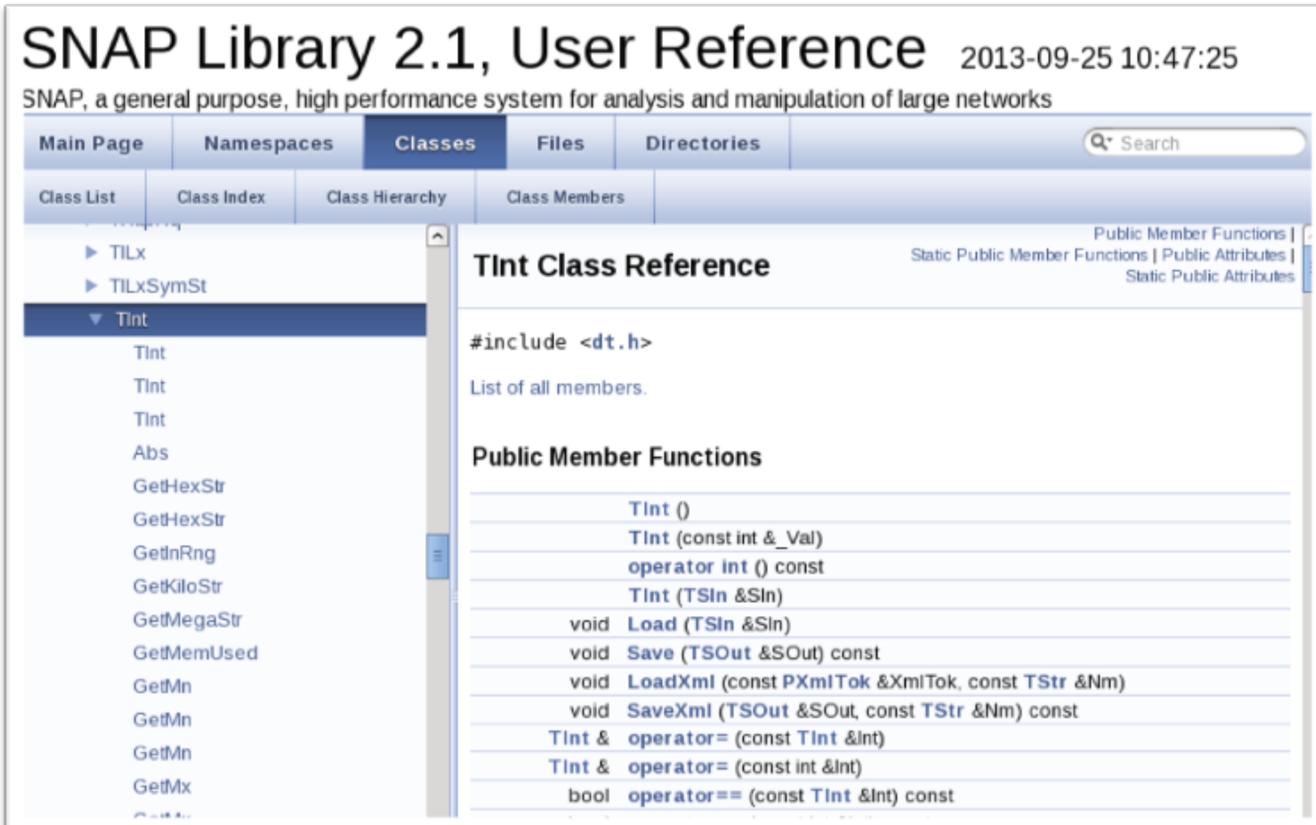
`GetKiloStr(val)`
A static method that returns the int *val* as a kilo-formatted string. If *val* is less than 1000, it returns *val* as a string. If *val* is greater than or equal to 1000, it returns a string in form of 'x.yK', where x is some digit from 1-9 and y from 0-9.

`GetMegaStr(val)`
A static method that returns the int *val* as a mega-formatted string. If *val* is less than 1000000, it returns the equivalent of *GetKiloStr(val)*. If *val* is greater than or equal to 1000000, it returns a string in the form of 'x.yM', where x is some digit from 1-9 and y from 0-9.

SNAP C++ Documentation

SNAP User Reference Manual

http://snap.stanford.edu/snap/doc.html



Vector Types

- Sequences of values of the same type
  - New values can be added the end
  - Existing values can be accessed or changed
- Naming convention: `T<type_name>V`
  - Examples: `TIntV, TFltV, TStrV`
- Common operations:
  - **`Add(<value>)`**: add a value
  - **`Len()`**: vector size
  - **`[<index>]`**: get or set a value of an existing element
  - **`for i in V:`** iteration over the elements

Vector Example

```
v = snap.TIntV()

v.Add(1)
v.Add(2)
v.Add(3)
v.Add(4)
v.Add(5)

print v.Len()

print v[3]
v[3] = 2*v[2]
print v[3]

for item in v:
    print item
for i in range(0, v.Len()):
    print i, v[i]
```

Create an empty vector

Add elements

Print vector size

Get and set element value

Print vector elements

Hash Table Types

- **A set of (key, value) pairs**
  - Keys must be of the same types, values must be of the same type (could be different from the key type)
  - New (key, value) pairs can be added
  - Existing values can be accessed or changed via a key
- **Naming: `T<key_type><value_type>H`**
  - **Examples:** `TIntStrH`, `TIntFltH`, `TStrIntH`
- **Common operations:**
  - **`[<key>]`**: add a new or get or set an existing value
  - **`Len()`**: hash table size
  - **`for k in H`**: iteration over keys
  - **`BegI()`, `IsEnd()`, `Next()`**: element iterators
  - **`GetKey(<i>)`**: get i-th key
  - **`GetDat(<key>)`**: get value associated with a key

Hash Table Example

```
h = snap.TIntStrH()

h[5] = “apple"
h[3] = “tomato"
h[9] = “orange"
h[6] = “banana"
h[1] = “apricot"

print h.Len()

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

h[3] = “peach"
print "h[3] =", h[3]

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

Create an empty table

Add elements

Print table size

Get element value

Set element value

Print table elements

Hash Tables: KeyID

- `T<key_type><value_type>H`
  - **`Key`:** item key, provided by the caller
  - **`Value`:** item value, provided by the caller
  - **`KeyId`:** integer, unique slot in the table, calculated by SNAP

KeyId	0	2	5
Key	100	89	95
Value	"David"	"Ann"	"Jason"

Pair Types

- **A pair of (value1, value2)**
  - Two values, type of value1 could be different from the value2 type
  - Existing values can be accessed
- **Naming:** **`T<type1><type2>Pr`**
  - **Examples:** `TIntStrPr`, `TIntFltPr`, `TStrIntPr`
- **Common operations:**
  - **`GetVal1`**: get value1
  - **`GetVal2`**: get value2

Pair Example

```
>>> p = snap.TIntStrPr(1,"one")
```

Create a pair

```
>>> print p.GetVal1()
1
>>> print p.GetVal2()
one
```

Print pair values

- **`TIntStrPrV`**: a vector of (integer, string) pairs
- **`TIntPrV`**: a vector of (integer, integer) pairs
- **`TIntPrFltH`**: a hash table with (integer, integer) pair keys and float values

Basic Graph and Network Classes

- **Graphs vs. Networks Classes:**
  - **`TUNGraph`**: undirected graph
  - **`TNGraph`**: directed graph
  - **`TNEANet`**: multigraph with attributes on nodes and edges
- Object types start with **P…**, since they use wrapper classes for garbage collection
  - **`PUNGraph, PNGraph, PNEANet`**
- Guideline
  - For class methods (functions) use **T**
  - For object instances (variables) use **P**

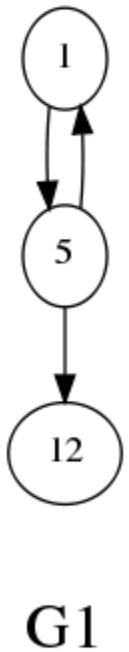
Graph Creation

```
G1 = snap.TNGraph.New()

G1.AddNode(1)
G1.AddNode(5)
G1.AddNode(12)

G1.AddEdge(1,5)
G1.AddEdge(5,1)
G1.AddEdge(5,12)

G2 = snap.TUNGraph.New()
N1 = snap.TNEANet.New()
```

Create directed graph

Add nodes before adding edges

Create undirected graph, directed network



Graph Traversal

Traverse nodes

```
for NI in G1.Nodes():
    print "node id %d, out-degree %d, in-degree %d"
      % (NI.GetId(), NI.GetOutDeg(), NI.GetInDeg())
```

Traverse edges

```
for EI in G1.Edges():
    print "(%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())
```

Traverse edges by nodes

```
for NI in G1.Nodes():
    for DstNId in NI.GetOutEdges():
        print "edge (%d %d)" % (NI.GetId(), DstNId)
```

Graph Saving and Loading

Save text

```
snap.SaveEdgeList(G4, "test.txt", “List of edges")
```

Load text

```
G5 = snap.LoadEdgeList(snap.PNGraph,"test.txt",0,1)
```

Save binary

```
FOut = snap.TFOut("test.graph")
G2.Save(FOut)
FOut.Flush()
```

Load binary

```
FIn = snap.TFIn("test.graph")
G4 = snap.TNGraph.Load(FIn)
```

Text File Format

- **Example file:** `wiki-Vote.txt`
  - Download from http://snap.stanford.edu/data

```
# Directed graph: wiki-Vote.txt
# Nodes: 7115 Edges: 103689
# FromNodeId    ToNodeId
0       1
0       2
0       3
0       4
0       5
2       6
…
```

Load text

```
G5 = snap.LoadEdgeList(snap.PNGraph,"test.txt",0,1)
```

Plotting in Snap.py

- **Plotting graph properties**
  - Gnuplot: http://www.gnuplot.info
- **Visualizing graphs**
  - Graphviz: http://www.graphviz.org
- **Other options**
  - Matplotlib: http://www.matplotlib.org

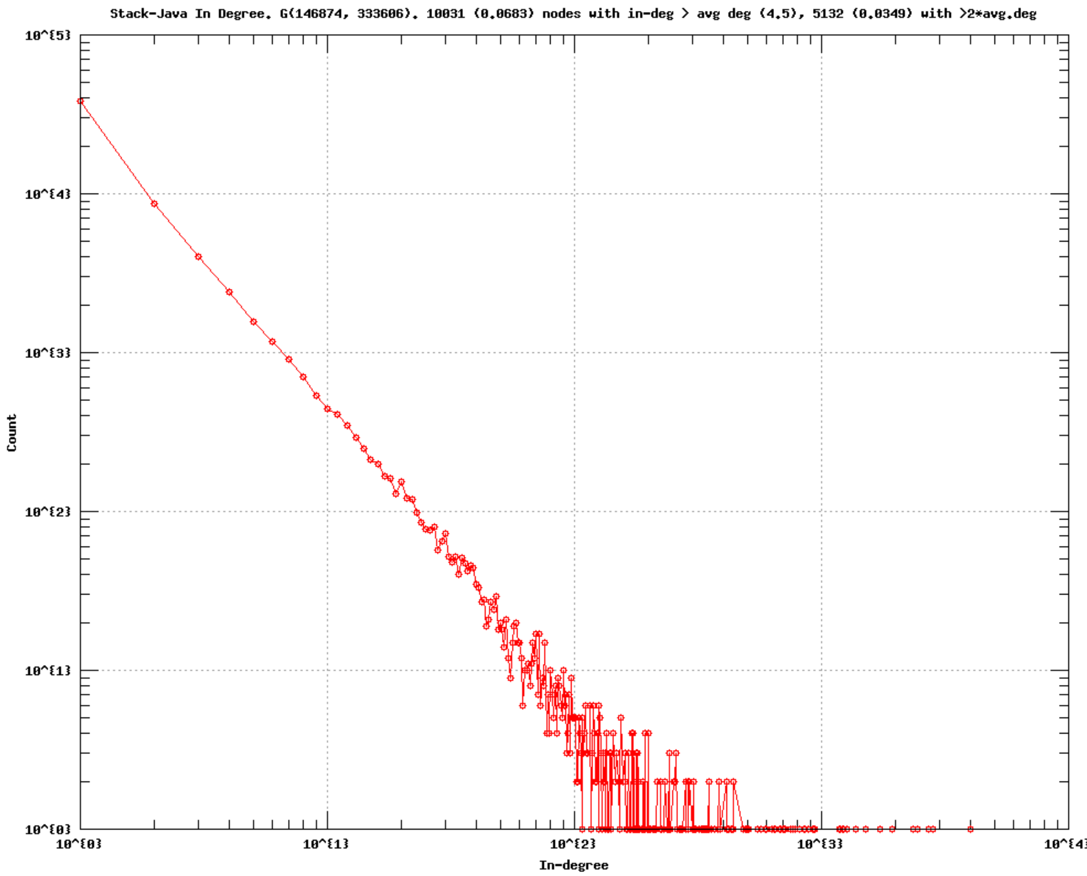
Plotting with Snap.py

- **Install Gnuplot:** http://www.gnuplot.info/

- Make sure that the directory containing wgnuplot.exe (for Windows) or gnuplot (for Linux, Mac OS X) is in your environmental variable **$PATH**

Plotting with Snap.py

```
G = snap.LoadEdgeList(snap.PNGraph, “stackoverflow-Java.txt", 0, 1)
snap.PlotInDegDistr(G, "Stack-Java", "Stack-Java In Degree")
```



Graph of Java QA on StackOverflow: in-degree distribution

Snap.py: Gnuplot Files

- **Snap.py** generates three files:
  - **.png** is the plot
  - **.tab** file contains the data (tab separated file)
  - **.plt** file contains the plotting commands

Drawing Graphs

- **Install GraphViz:** http://www.graphviz.org/
- Make sure that the directory containing GraphViz is in your environmental variable **$PATH**

Drawing Graphs with Snap.py

```
G1 = snap.TNGraph.New()

G1.AddNode(1)
G1.AddNode(5)
G1.AddNode(12)

G1.AddEdge(1,5)
G1.AddEdge(5,1)
G1.AddEdge(5,12)
```

Create graph

G1

```
NIdName = snap.TIntStrH()
NIdName[1] = "1"
NIdName[5] = "5"
NIdName[12] = "12"
```

Set node labels

Draw

```
snap.DrawGViz(G1, snap.gvlDot, "G1.png", "G1", NIdName)
```

Print Graph Information

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

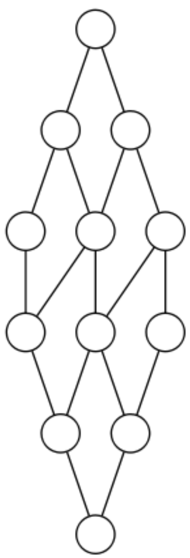
Output:

```
QA Stats: Directed
  Nodes:                      146874
  Edges:                      333606
  Zero Deg Nodes:             0
  Zero InDeg Nodes:           83443
  Zero OutDeg Nodes:          30963
  NonZero In-Out Deg Nodes:   32468
  Unique directed edges:      333606
  Unique undirected edges:    333481
  Self Edges:                 20600
  BiDir Edges:                20850
  Closed triangles:           41389
  Open triangles:             51597174
  Frac. of closed triads:     0.000802
  Connected component size:   0.893201
  Strong conn. comp. size:    0.029433
  Approx. full diameter:      14
  90% effective diameter:     5.588639
```

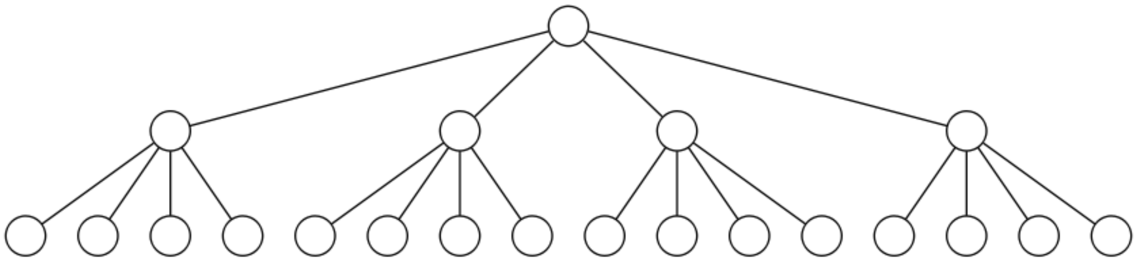
Basic Graph Generators

- **Complete, circle, grid, star, tree graphs**

```
GG = snap.GenGrid(snap.PUNGraph, 4, 3)
GT = snap.GenTree(snap.PUNGraph, 4, 2)
```



G-4-3

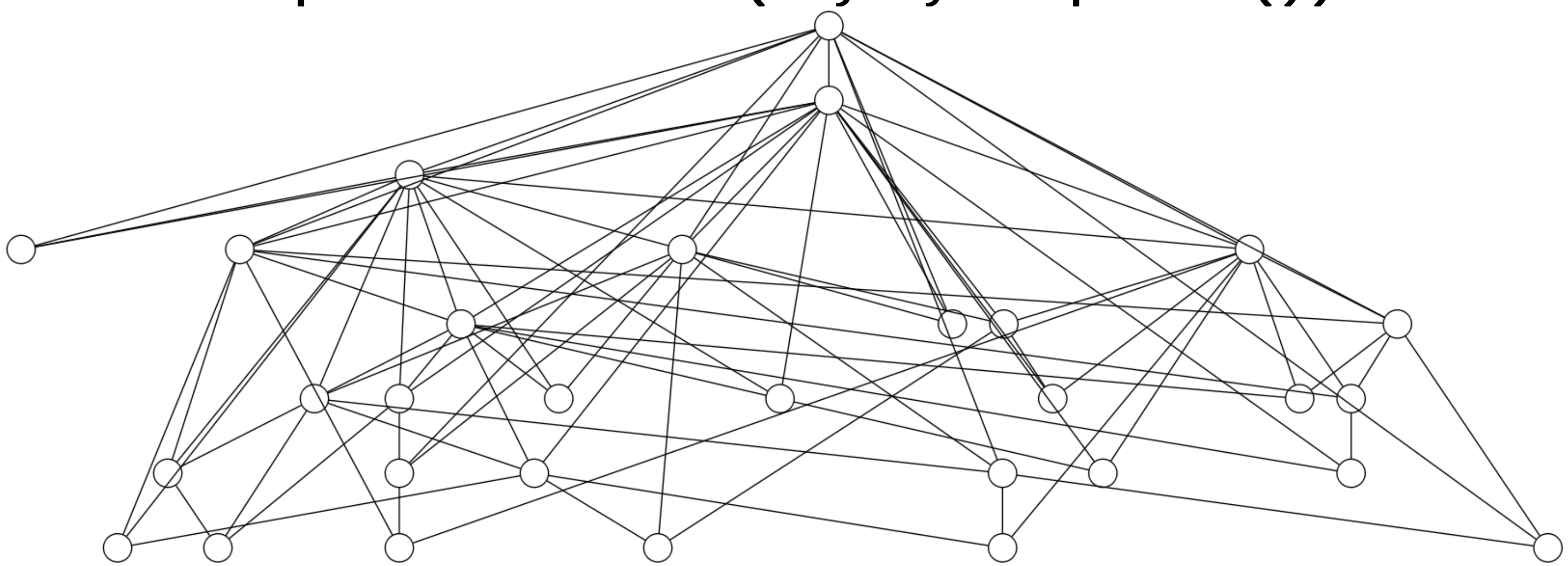


T-4-2

Advanced Graph Generators

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

```
GPA = snap.GenPrefAttach(30, 3, snap.TRnd())
```



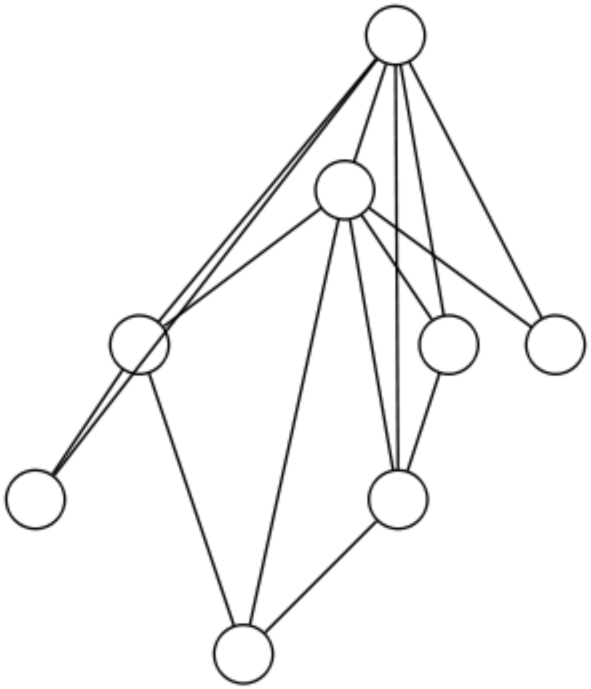
PA-30

Subgraphs and Conversions

- Extract subgraphs, convert from one graph type to another

Get an induced subgraph on a set of nodes NIdV:

```
NIdV = snap.TIntV()
for i in range(1,9): NIdV.Add(i)

SubGPA = snap.GetSubGraph(GPA, NIdV)
```



SPA-8

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

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

Get largest WCC

```
WccV = snap.TIntPrV()
snap.GetWccSzCnt(G, WccV)
```

Get WCC sizes

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

Node Degrees

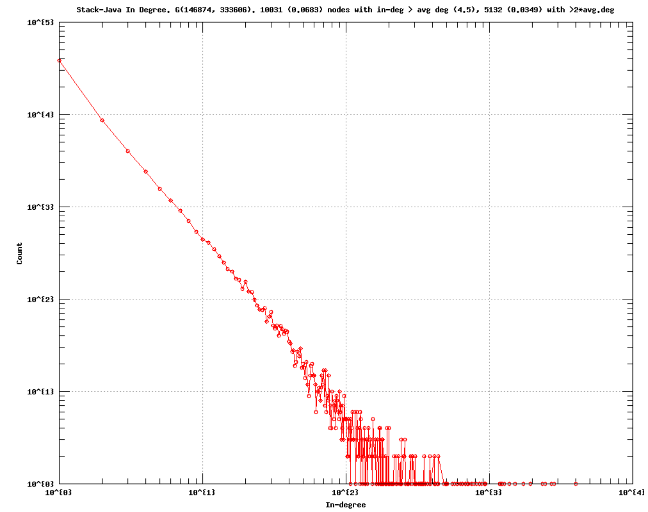
- **Analyze node connectivity**
  - Find node degrees, maximum degree, degree distribution
  - In-degree, out-degree, combined degree

```
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())
```

Get node with max degree

Get degree distribution

```
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

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

Calculate node PageRank scores

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

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
```

Count triads

```
CC = snap.GetClustCf(GPA)
print "clustering coefficient", CC
```

Calculate clustering coefficient

Breadth and Depth First Search

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

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

Calculate diameter

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

Calculate effective diameter

Community Detection

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

Clauset-Newman-Moore

```
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
```

Spectral properties of a graph

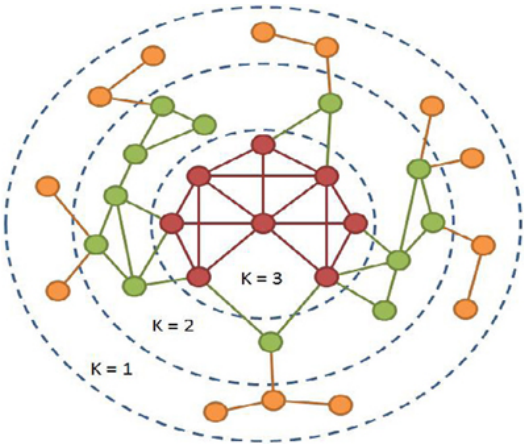
- **Calculations based on graph adjacency matrix**
  - Get Eigenvalues, Eigenvectors
  - Get Singular values, leading singular vectors

```
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




```
Core3 = snap.GetKCore(G, 3)
```

Calculate 3-core