# An Introduction to Snap.py

## SNAP for Python

Author: Rok Sosic

Created: Sep 26, 2013

# Content

- Introduction to Snap.py
- Tutorial
- Plotting
- Q&A

# What is SNAP?

- **S**tanford **N**etwork **A**nalysis **P**roject (SNAP)
- General purpose, high performance system for analysis and manipulation of large networks
- Scales to massive networks with hundreds of millions of nodes, and billions of edges
- Manipulates large networks, calculates structural properties, generates graphs, and supports attributes on nodes and edges
- Software is C++ based
- Web site at http://snap.stanford.edu

# What is Snap.py？

- Snap.py: SNAP for Python
  - Provides SNAP functionality in Python
- C++
  - Good - fast program execution
  - Downside - complex language, needs compilation
- Python
  - Downside – slow program execution
  - Good – simple language, interactive use
- Snap.py
  - Good – fast program execution
  - Good – simple language, interactive use
- Web site at http://snap.stanford.edu/snap/snap.py.html

# Snap.py Documentation

- Check out Snap.py at:

  http://snap.stanford.edu/snap/snap.py.html
  - Packages for Mac OS X, Windows, Linux
  - Quick Introduction and Tutorial
- SNAP documentation (snap.stanford.edu)
  - User Reference Manual
    - Top level graph classes **TUNGraph**, **TNGraph**, **TNEANet**
    - Namespace **TSnap**
  - Developer resources
    - Developer Reference Manual
    - GitHub repository
    - SNAP C++ Programming Guide

# Snap.py Installation

- Download the Snap.py package for your platform: http://snap.stanford.edu/snap/snap.py.html
  - Packages for Mac OS X, Windows, Linux (CentOS)
  - 64-bit only – OS, Python
  - Mac OS X, 10.7.5 or later
  - Windows, install Visual Studio
  - Snap.py is beta version, report problems
- Installation
  - Follow instructions on the page above
  - Check out Piazza for troubleshooting, non-standard configurations
- Alternatively, use **corn.stanford.edu**

# Content

- Introduction to Snap.py
- **Tutorial**
- Plotting
- Q&A

# Snap.py Tutorial

- On the Web: http://snap.stanford.edu/snappy/doc/tutorial/index-tut.html
- Basic types
- Vectors, hash tables and pairs
- Graphs and networks
- Graph creation
- Adding and traversing nodes and edges
- Saving and loading graphs
- Graph manipulation
- Computing structural properties

# Important Background

- Always import **snap** module

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

# Basic Types in Snap.py and SNAP

- **TInt**: int
- **TFlt**: float
- **TStr**: str
  - Note: do not use an empty string “” in parameters
- Automatically converted between C++ and Python
  - In general no need to deal with basic types explicitly
- An example (just for illustration)

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

# SNAP C++ Documentation

- Snap.stanford.edu
  - User Reference Documentation

## SNAP Library 2.1, User Reference 2013-09-25 10:47:25

SNAP, a general purpose, high performance system for analysis and manipulation of large networks

Main Page | Namespaces | Classes | Files | Directories | Search

Class List | Class Index | Class Hierarchy | Class Members

- TILx
- TILxSymSt
- TInt
  - TInt
  - TInt
  - TInt
  - Abs
  - GetHexStr
  - GetHexStr
  - GetInRng
  - GetKiloStr
  - GetMegaStr
  - GetMemUsed
  - GetMn
  - GetMn
  - GetMn
  - GetMx
  - GetMx

Public Member Functions | Static Public Member Functions | Public Attributes | Static Public Attributes

### TInt Class Reference

```
#include <dt.h>
```

List of all members.

#### Public Member Functions

| | |
|---|---|
| | TInt () |
| | TInt (const int &_Val) |
| | operator int () const |
| | TInt (TSIn &SIn) |
| void | Load (TSIn &SIn) |
| void | Save (TSOut &SOut) const |
| void | LoadXml (const PXmlTok &XmlTok, const TStr &Nm) |
| void | SaveXml (TSOut &SOut, const TStr &Nm) const |
| TInt & | operator= (const TInt &Int) |
| TInt & | operator= (const int &Int) |
| bool | operator== (const TInt &Int) const |
| bool | operator== (const int &Int) const |

# Vector Types

- Sequences of values of the same type
  - New values can be added the end
  - Existing values can be accessed or changed
- Naming: **<type_name>V**
  - TIntV, TFltV, TStrV
- Operations:
  - **Add**, add a value
  - **Len**, vector size
  - **[]**, get a value of an existing element
  - **SetVal()**, set a value of an existing element
  - **for i in v**, iterator

# 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[2]
v.SetVal(2, 2*v[2])
print v[2]

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

# 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: **<type1><type2>H**
  - TIntStrH, TIntFltH, TStrIntH
- Operations:
  - **AddDat**, add a new or change an existing value
  - **Len**, table size
  - **GetDat**, get a value of an existing element
  - **for i in h**, iterator
    - **GetKey** get key, **GetDat** get value

# Hash Table Example

```
h = snap.TIntStrH()

h.AddDat(5,"five")
h.AddDat(3,"three")
h.AddDat(9,"nine")
h.AddDat(6,"six")
h.AddDat(1,"one")

print h.Len()

print "h[3] =", h.GetDat(3)

h.AddDat(3,"four")
print "h[3] =", h.GetDat(3)

for item in h:
    print item.GetKey(), item.GetDat()
```

# 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: **<type1,type2>Pr**
  - TIntStrPr, TIntFltPr, TStrIntPr
- Operations:
  - **GetVal1**, get value1
  - **GetVal2**, get value2

# Pair Example

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

print p.GetVal1()
print p.GetVal2()
```

- **TIntPrFltH**, a hash table with (integer, integer) pair keys and float values

# Basic Graph and Network Types

- **TUNGraph**: undirected graph
- **TNGraph**: directed graph
- **TNEANet**: multigraph with attributes on nodes and edges
- Pointers to graphs, names start with **P**
  - **PUNGraph, PNGraph, PNEANet**
  - for class methods (functions) use **T**
  - for instances (variables) use **P**

# Graph Creation

```
G1 = snap.TUNGraph.New()
G2 = snap.TNGraph.New()
N1 = snap.TNEANet.New()

G1.AddNode(1)
G1.AddNode(5)
G1.AddNode(32)

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

- Add nodes before edges

# Traversal

```
# create a directed random graph on 100 nodes and 1k edges
G2 = snap.GenRndGnm(snap.PNGraph, 100, 1000)

# traverse the nodes
for NI in G2.Nodes():
    print "node id %d, out-degree %d, in-degree %d" % (
        NI.GetId(), NI.GetOutDeg(), NI.GetInDeg())

# traverse the edges
for EI in G2.Edges():
    print "(%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())

# traverse the edges by nodes
for NI in G2.Nodes():
    for Id in NI.GetOutEdges():
        print "edge (%d %d)" % (NI.GetId(), Id)
```

# Saving and Loading

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

# load binary
FIn = snap.TFIn("test.graph")
G4 = snap.TNGraph.Load(FIn)

# save and load from a text file
snap.SaveEdgeList(G4, "test.txt", “List of edges")
G5 = snap.LoadEdgeList(snap.PNGraph, "test.txt", 0, 1)
```

# Edge List, Text File Format

- Example file, **wiki-Vote.txt**

```
# Directed graph: wiki-Vote.txt
# Nodes: 7115 Edges: 103689
# FromNodeId    ToNodeId
0       1
0       2
0       3
0       4
0       5
2       6
2       7
2       8
…
```

# Graph Manipulations

```
# create a directed random graph on 10k nodes and 5k edges
G6 = snap.GenRndGnm(snap.PNGraph, 10000, 5000)

# convert to undirected graph
G7 = snap.ConvertGraph(snap.PUNGraph, G6)

# get largest weakly connected component
WccG = snap.GetMxWcc(G6)

# generate a network using Forest Fire model
G8 = snap.GenForestFire(1000, 0.35, 0.35)

# get a subgraph induced on nodes {0,1,2,3,4}
SubG = snap.GetSubGraph(G8, snap.TIntV.GetV(0,1,2,3,4))

# get 3-core of G8
Core3 = snap.GetKCore(G8, 3)

# delete nodes of out degree 3 and in degree 2
snap.DelDegKNodes(G8, 3, 2)
```

# Structural Properties

```
# create a directed random graph on 10k nodes and 1k edges
G9 = snap.GenRndGnm(snap.PNGraph, 10000, 1000)

# define a vector of pairs of integers (size, count) and
# get a distribution of connected components (component size, count)
CntV = snap.TIntPrV()
snap.GetWccSzCnt(G9, CntV)
for p in CntV:
    print "size %d: count %d" % (p.GetVal1(), p.GetVal2())

# generate a Preferential Attachment graph 100 nodes, out-degree of 3
G10 = snap.GenPrefAttach(100, 3)

# define a vector of floats and get first eigenvector of
# graph adjacency matrix
EigV = snap.TFltV()
snap.GetEigVec(G10, EigV)
nr = 0
for f in EigV:
    nr += 1
    print "%d: %.6f" % (nr, f)
```

# Content

- Introduction to Snap.py
- Tutorial
- **Plotting**
- Q&A

# Plotting Options in Snap.py

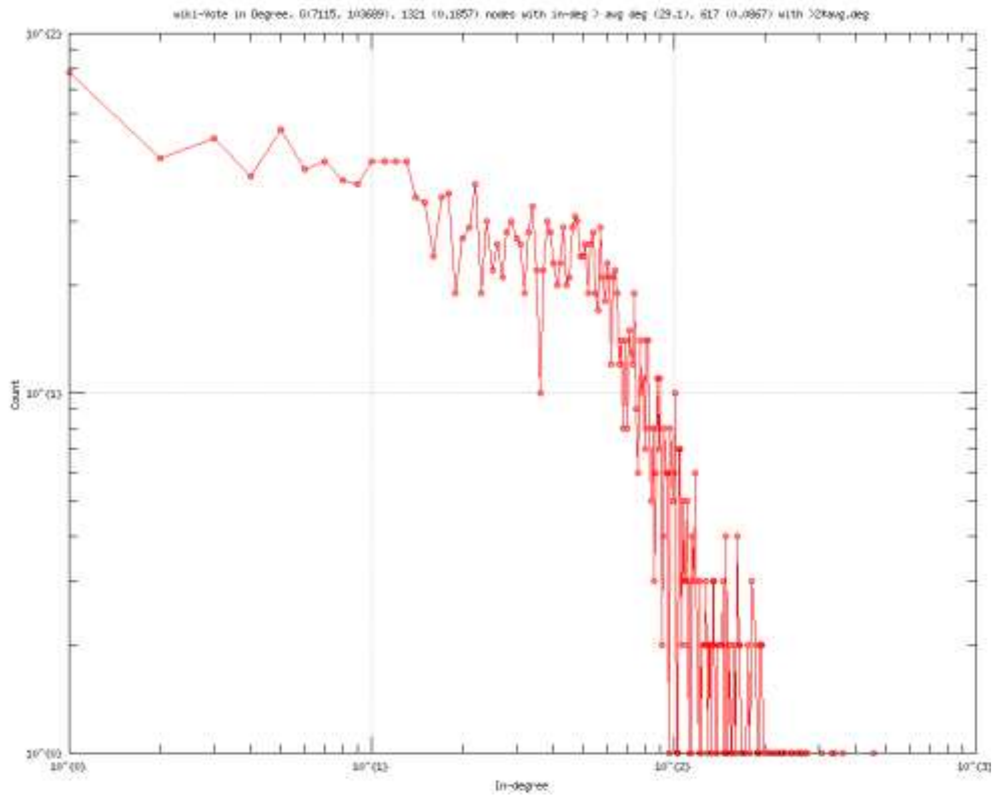
- Plotting graph properties
  - Gnuplot: http://www.gnuplot.info
- Visualizing graphs
  - Graphviz: http://www.graphviz.org
- Other options
  - Matplotlib: http://www.matplotlib.org

# Plotting with Snap.py

- Install Gnuplot from 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

```
# Produce a plot of the in-degree node distribution
import snap
G = snap.LoadEdgeList(snap.PNGraph, "wiki-Vote.txt", 0, 1)
snap.PlotInDegDistr(G, "wikiInDeg", "wiki-vote In Degree")
```



# Gnuplot

- After executing, three files generated


  - .png or .eps is the plot
  - .tab file contains the data
  - .plt file is the plotting command for gnuplot

# Visualize Your Graph

- Need to install GraphViz software
  http://www.graphviz.org/
- Add GraphViz path to environment variable

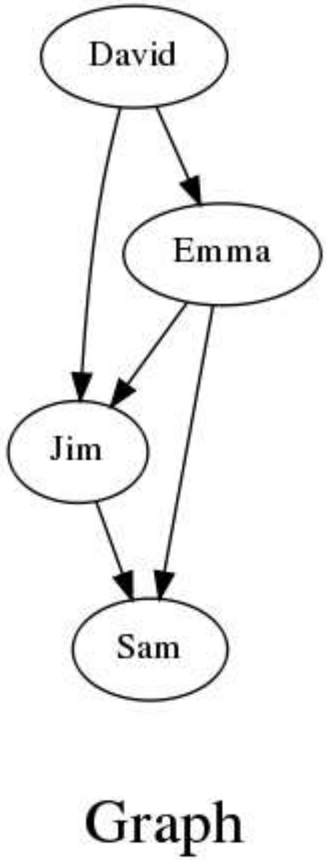
# Visualizing Graphs

```
# Visualize a directed graph
import snap

G = snap.TNGraph.New()
G.AddNode(1)
G.AddNode(2)
G.AddNode(3)
G.AddNode(4)
G.AddEdge(1,2)
G.AddEdge(2,3)
G.AddEdge(1,3)
G.AddEdge(2,4)
G.AddEdge(3,4)

S = snap.TIntStrH()
S.AddDat(1,"David")
S.AddDat(2,"Emma")
S.AddDat(3,"Jim")
S.AddDat(4,"Sam")

snap.DrawGViz(G, snap.gvlDot, "gviz.png", "Graph", S)
```



# Datasets In SNAP

- http://snap.stanford.edu/data/index.html
- Some examples:
  - **Social networks:** online social networks, edges represent interactions between people
  - **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
  - **Twitter and Memetracker :** Memetracker phrases, links and 467 million Tweets

# Conclusion

- Q&A
- Thank you!