Tutorial: Large Scale Network Analytics with SNAP

http://snap.stanford.edu/proj/snap-www

Rok Sosič, Jure Leskovec
Stanford University

WWW-15, Florence, Italy **May, 2015**

SNAP Tutorial: Content

- Motivation
- Introduction to SNAP
- Snap.py for Python
- Network analytics
- SNAP network datasets
- SNAP for C++
- Hands-on exercise

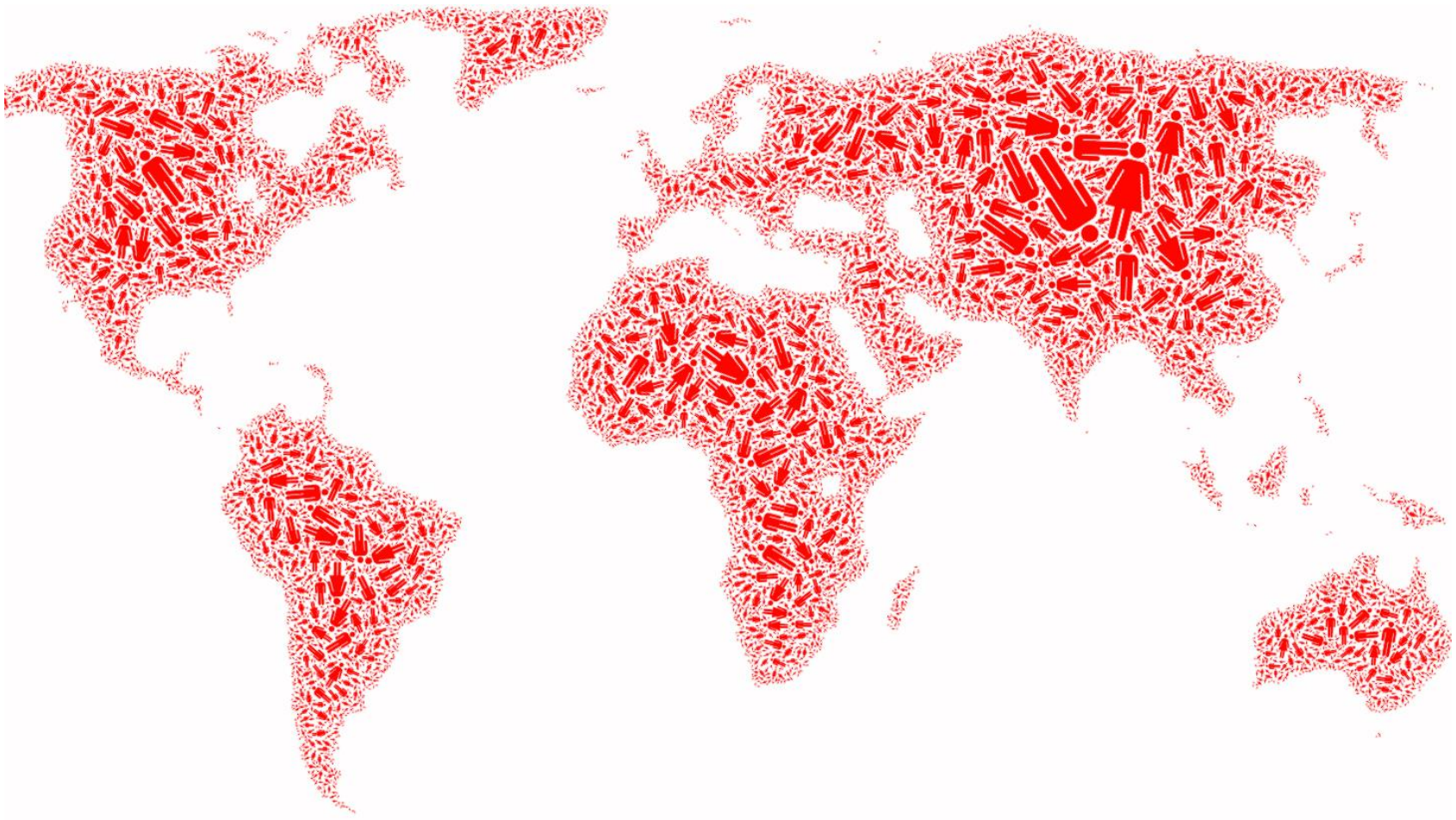
Contact information:
Rok Sosič, rok@cs.stanford.edu

Slides available at:
http://snap.stanford.edu/proj/snap-www

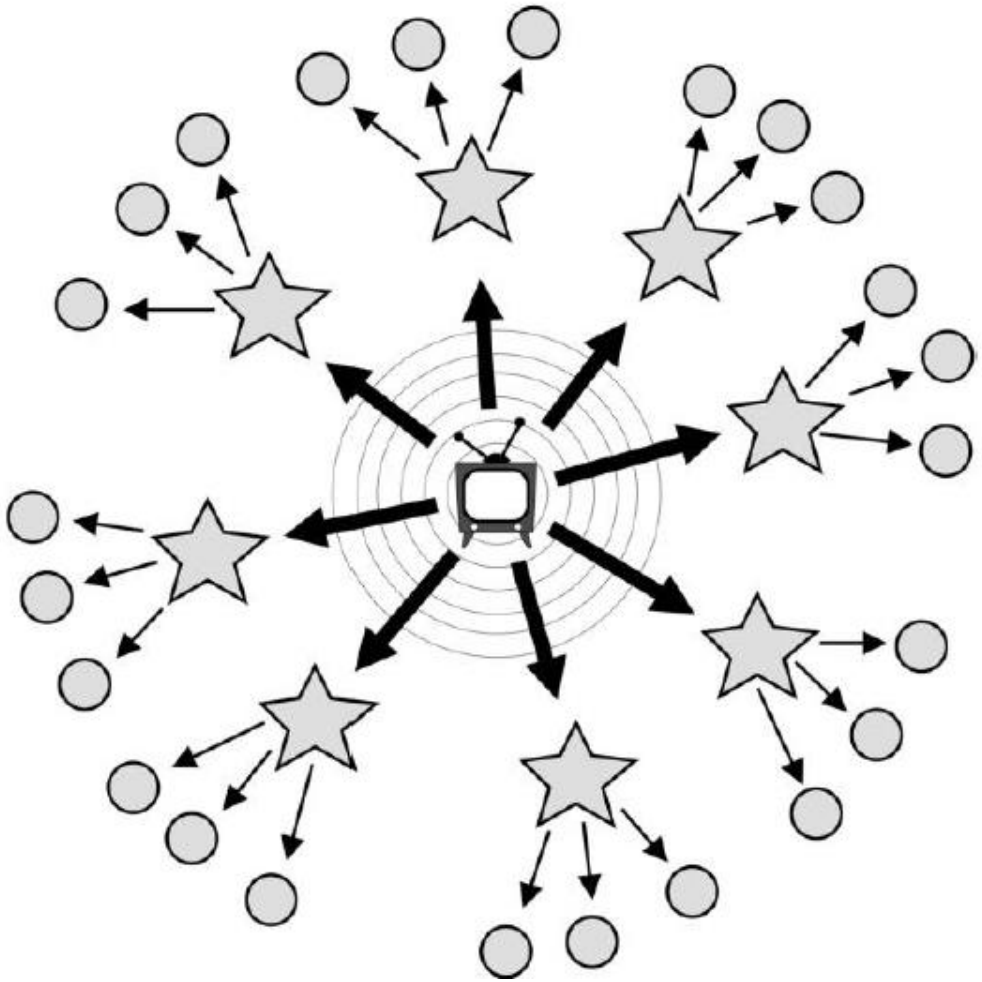
Why Networks?

Networks are a general language for describing complex systems

Friends & Family

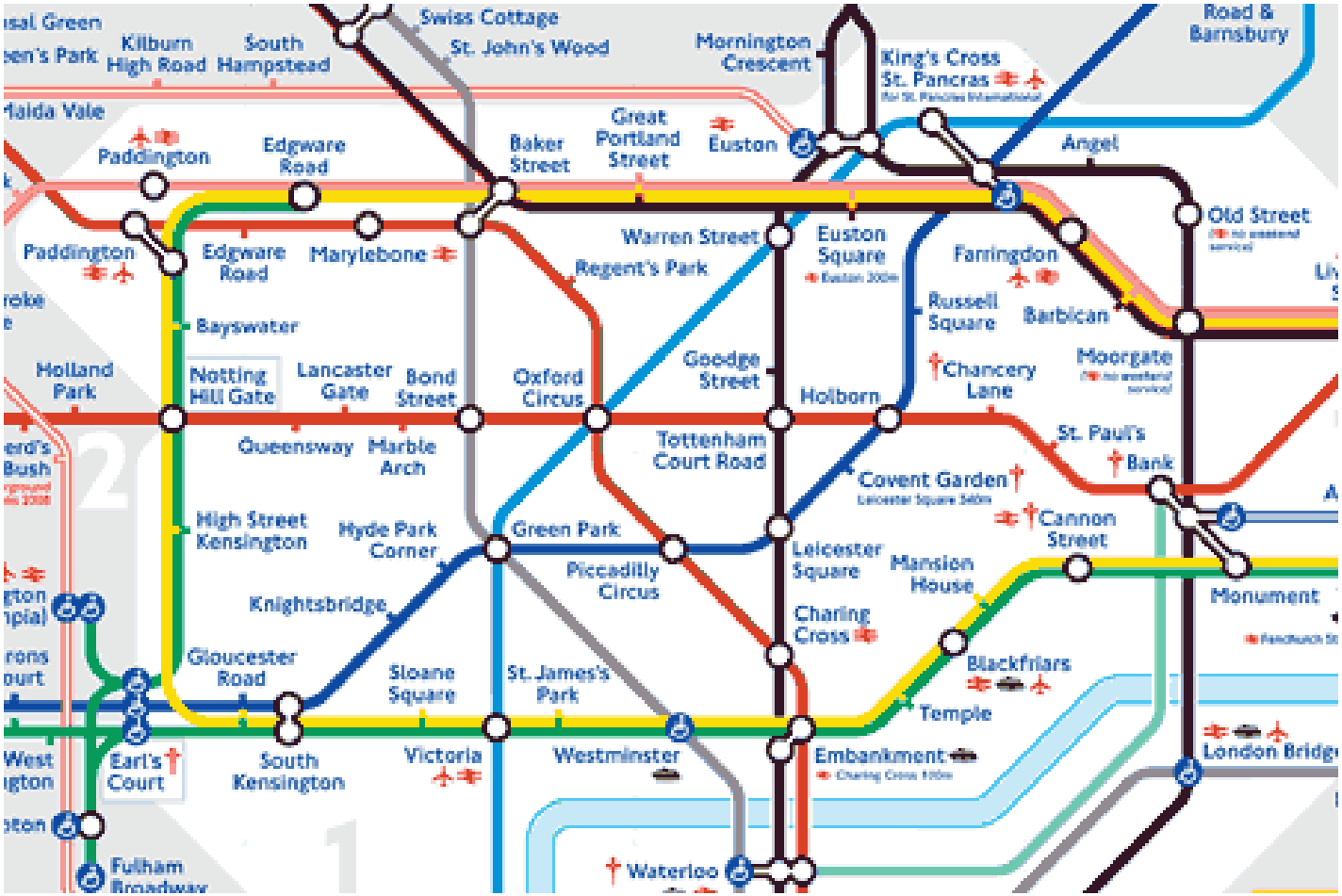


Society

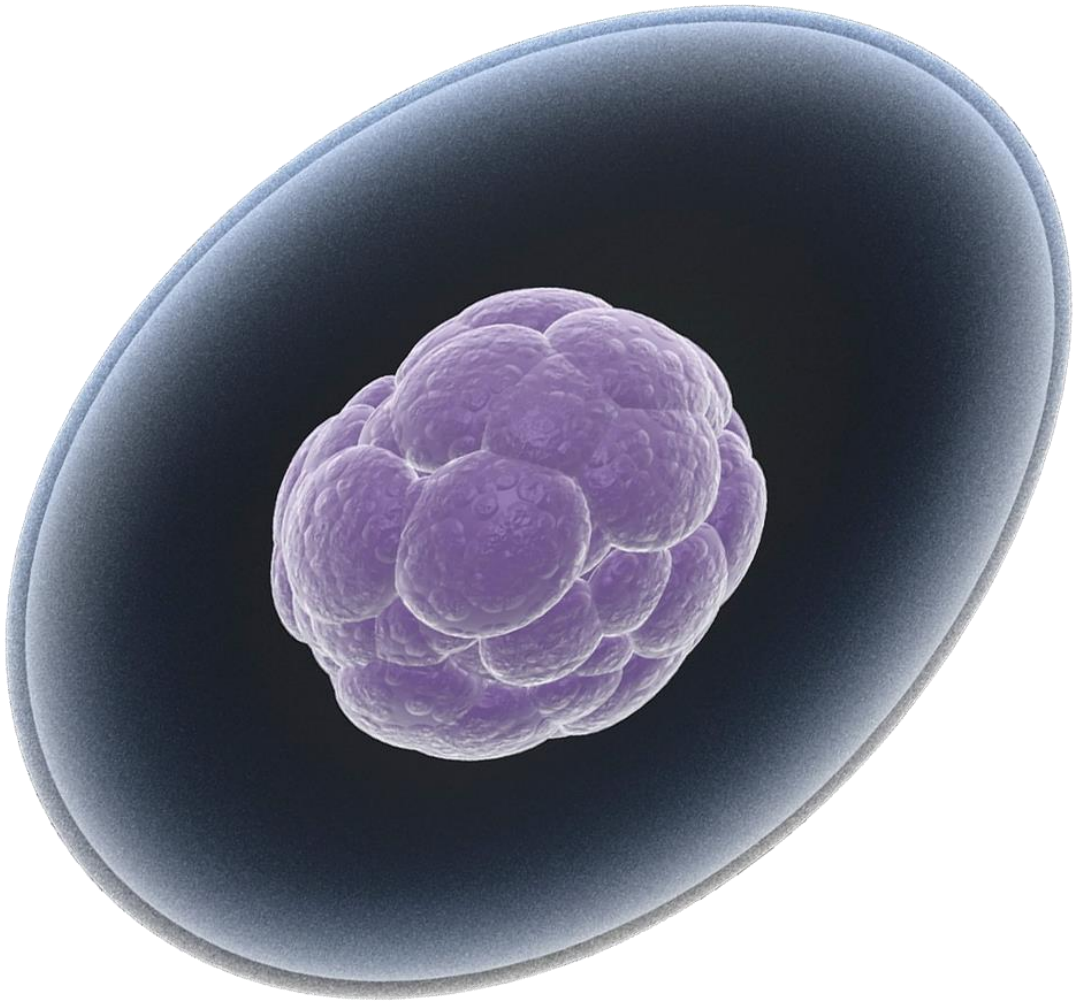


Media & Information

World economy

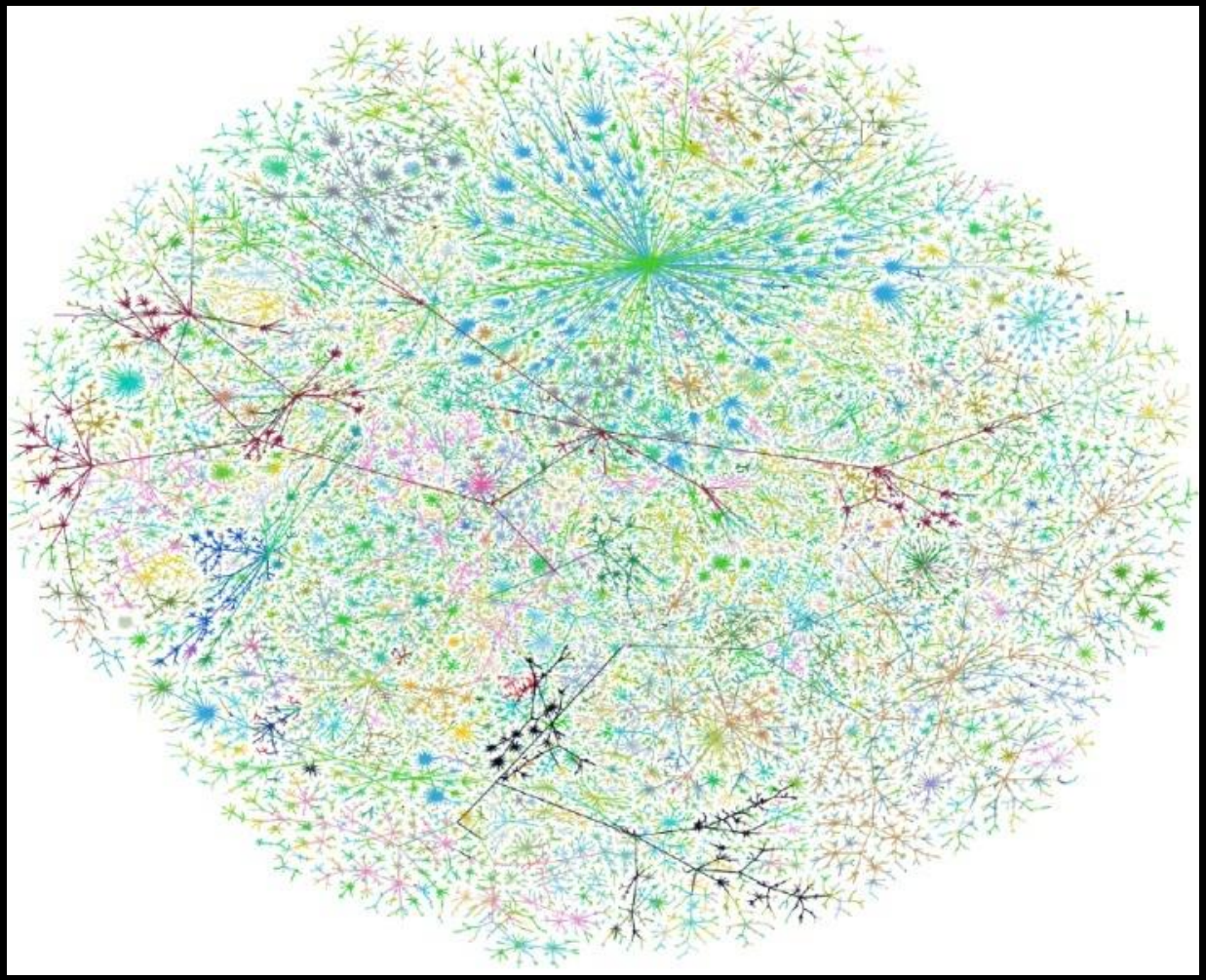

Swiss Cottage
St. John's Wood
Kilburn High Road
South Hampstead
Mornington Crescent
King's Cross St. Pancras
Road & Barnsbury
Paddington
Edgware Road
Baker Street
Great Portland Street
Euston
Angel
Old Street
Warren Street
Euston Square
Farringdon
Marylebone
Regent's Park
Russell Square
Barbican
Bayswater
Holland Park
Notting Hill Gate
Lancaster Gate
Bond Street
Oxford Circus
Goodge Street
Holborn
Chancery Lane
Moorgate
Queensway
Marble Arch
Tottenham Court Road
St. Paul's
Bank
Covent Garden
Cannon Street
High Street Kensington
Hyde Park Corner
Green Park
Piccadilly Circus
Leicester Square
Mansion House
Monument
Knightsbridge
Charing Cross
Gloucester Road
Sloane Square
St. James's Park
Blackfriars
Temple
Earl's Court
South Kensington
Victoria
Westminster
Embankment
London Bridge
Waterloo
Fulham Broadway

Roads



Human cell

Brain



Networks!

Introduction to SNAP

Rok Sosič, Jure Leskovec
Stanford University

WWW-15, Florence, Italy **May, 2015**

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

Graphs and Networks in SNAP

- **In the context of SNAP software**
  - **Graphs** consists of nodes and edges
    - An edge connects two points (or is a loop)
  - **Networks** are graphs where nodes and edges can have attributes (features, values)

- In presentation and documentation, terms "graph" and "network" are often used interchangeably to mean graph and/or network
  - Specific meaning is usually evident from the context

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 web and 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

Snap.py: SNAP for Python

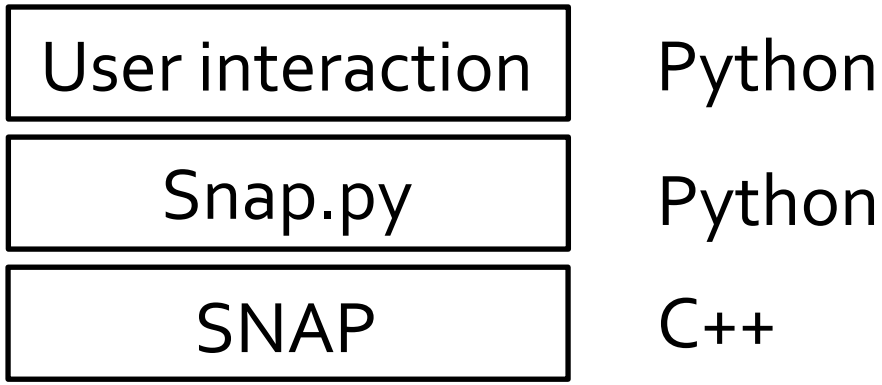
Rok Sosič, Jure Leskovec
Stanford University

WWW-15, Florence, Italy **May, 2015**

What is Snap.py?

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

http://snap.stanford.edu/snappy



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

Installing Snap.py

- **Requires Python 2.x**
  - **Download and install Python 2.x:**

    http://www.python.org
- **Download the Snap.py for your platform:**

    http://snap.stanford.edu/snappy
  - Packages for Mac OS X, Windows, Linux (CentOS)
    - OS must be 64-bit
    - Mac OS X, 10.7.5 or later
    - Windows, install Visual C++ Redistributable Runtime
      http://www.microsoft.com/en-us/download/details.aspx?id=30679
- **Installation:**
  - Follow instructions on the Snap.py webpage

```
python setup.py install
```

If you encounter problems, please report them to us or post to the mailing list

Snap.py: Important

- The most important step:

  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:

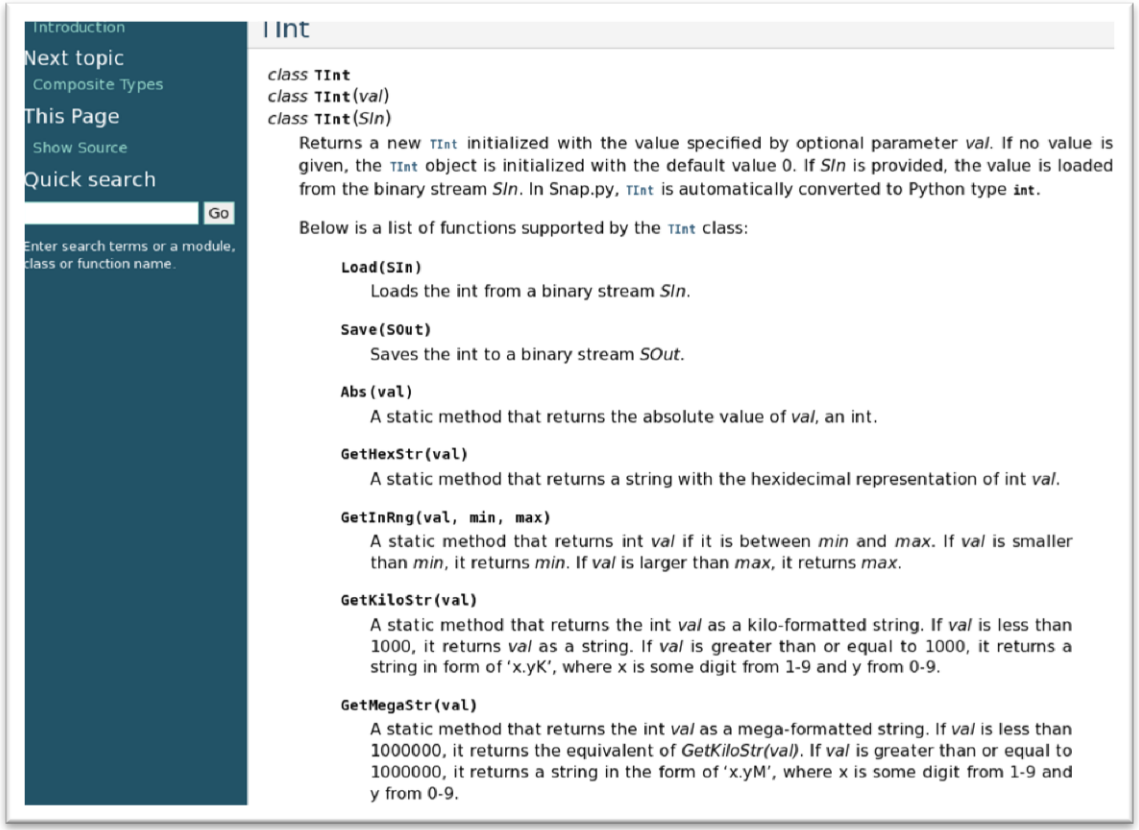
```
>>> i = snap.TInt(10)
>>> print i.Val
10
```

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

Snap.py Reference Documentation

For more information check out Snap.py Reference Manual

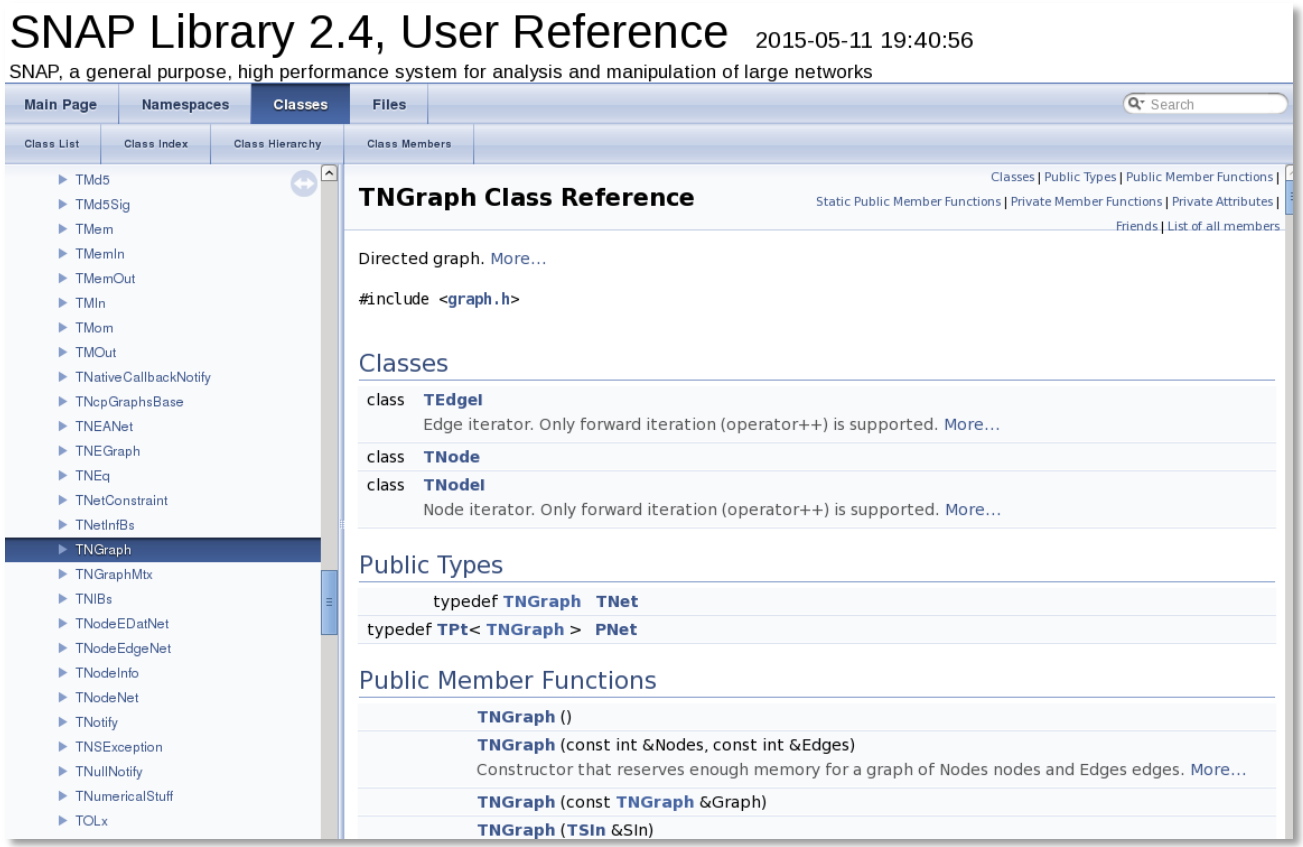
http://snap.stanford.edu/snappy/doc/reference/index-ref.html



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 at the end
  - Existing values can be accessed or changed
- **Naming convention: `T<value_type>V`**
  - Examples: `TIntV, TFltV, TStrV`
- **Common operations:**
  - **`Add(<value>)`**: append a value at the end
  - **`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
    - Value type can 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 value or get or set an existing value
  - **`Len()`**: hash table size
  - **`for k in H`**: iteration over keys

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()
```

Directed graph

Add nodes before adding edges

G1

Undirected graph, directed network

Graph Traversal

Node traversal

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

Edge traversal

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

Edge traversal by nodes

```
for NI in G1.Nodes():
    for DstNId in NI.GetOutEdges():
        print "(%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

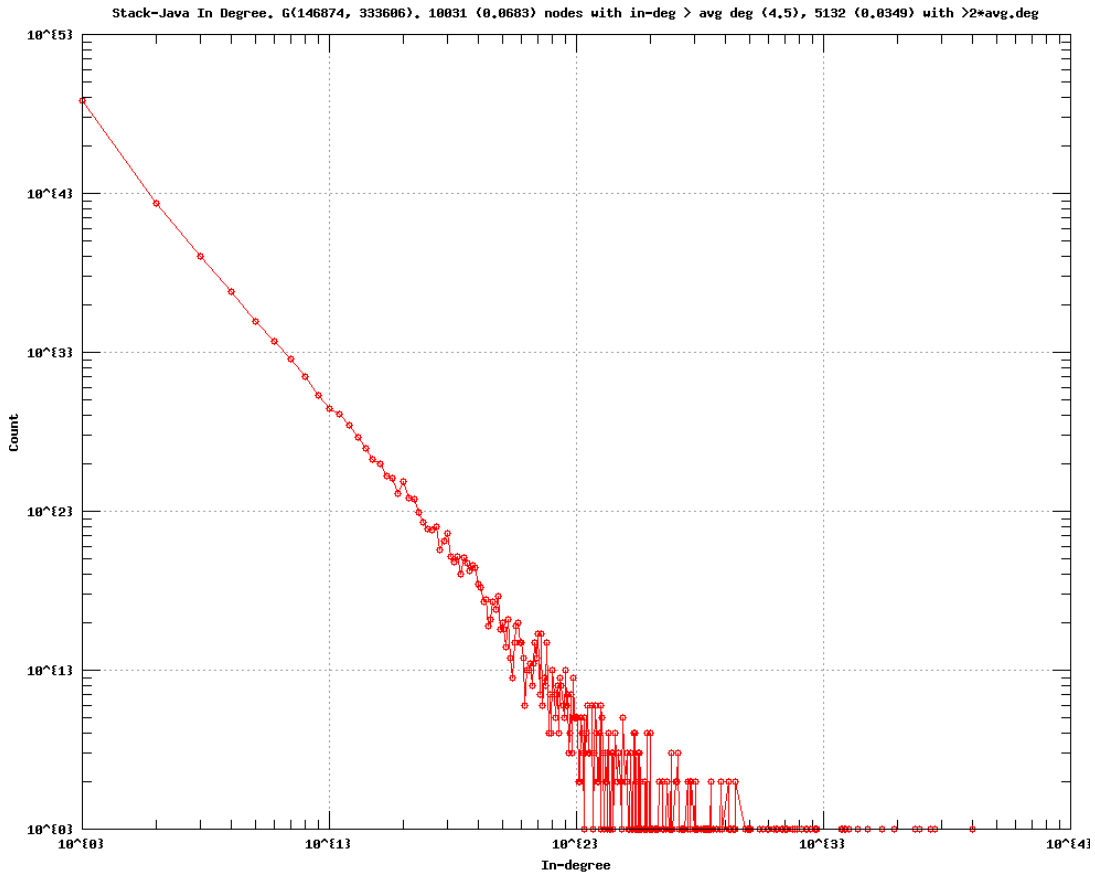
Plotting with Snap.py

- **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

```
import snap
G = snap.LoadEdgeList(snap.PNGraph, "qa.txt", 1, 5)
snap.PlotInDegDistr(G, "Stack-Java", "Stack-Java In Degree")
```



Graph of Java QA on StackOverflow: in-degree distribution

Snap.py + Gnuplot

- **Snap.py** generates three files:

  - **.png** or **.eps** is the plot
  - **.tab** file contains the data (tab separated file)
  - **.plt** file contains the plotting commands

Drawing Graphs

- **InstallGraphViz:** 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)
```

Network Analytics with SNAP

Rok Sosič, Jure Leskovec
Stanford University

WWW-15, Florence, Italy **May, 2015**

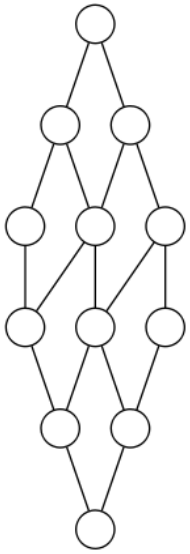
Overview of Network Analytics

- **How to get a network**
  - From a **real-world dataset**
  - Generate a **synthetic network**
  - From an **existing network**
- **Calculate network properties**
  - **Quick summary** of network properties
  - **Global connectivity**: connected components
  - **Local connectivity**: node degrees
  - **Key nodes** in the network: node centrality
  - **Neighborhood connectivity**: triads, clustering coefficient
  - **Graph traversal**: breadth and depth first search
  - **Groups of nodes**: community detection
  - **Global graph properties**: spectral graph analysis
  - **Core nodes**: K-core decomposition

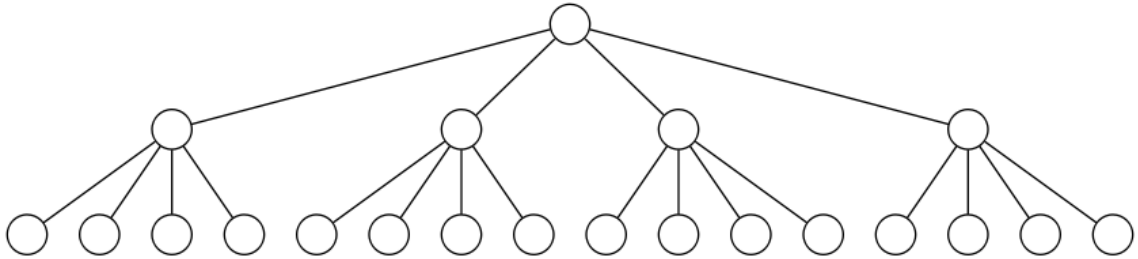
Basic Graph Generators

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

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



G-4-3

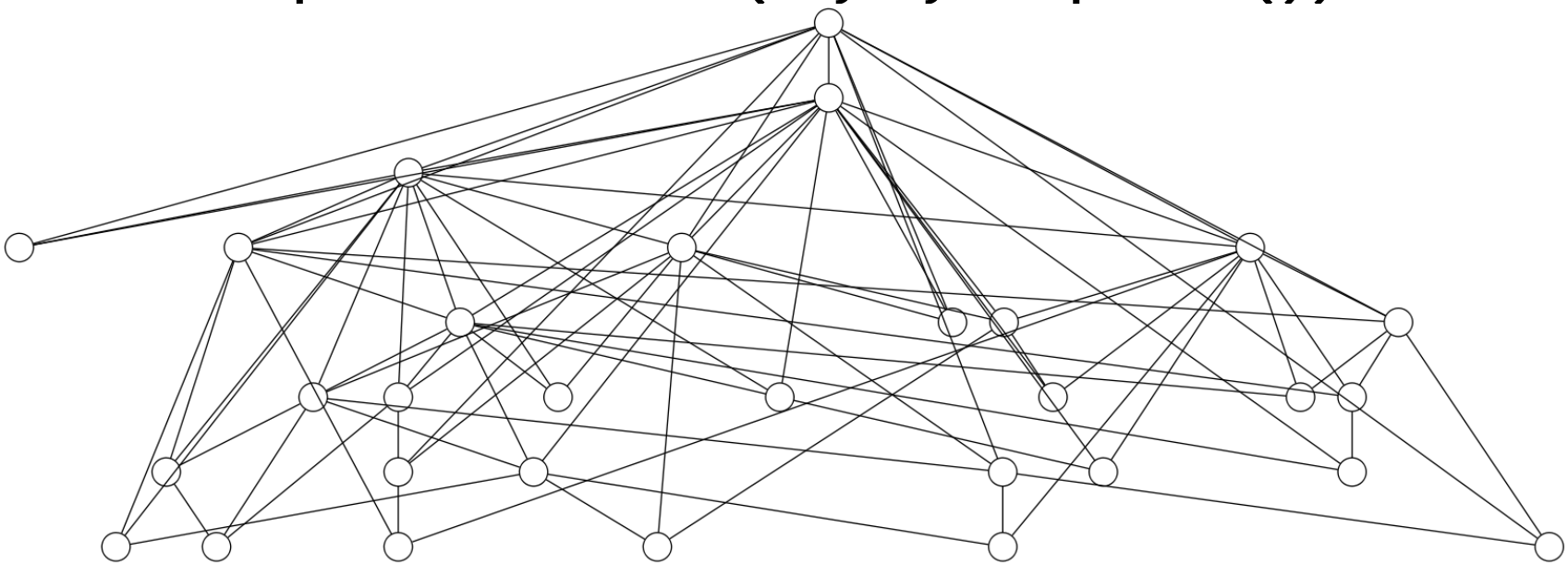


T-4-2

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

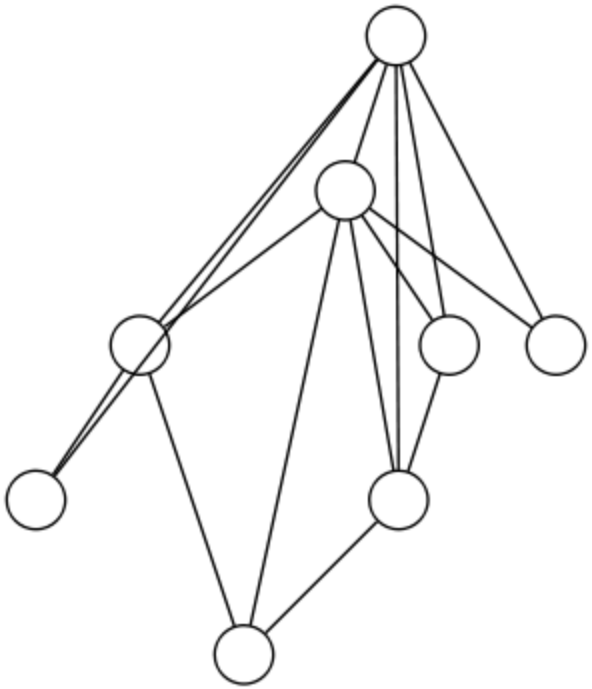
Subgraphs and Conversions

- 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

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:                     188406
  Edges:                     415174
  Zero Deg Nodes:            0
  Zero InDeg Nodes:          108618
  Zero OutDeg Nodes:         38319
  NonZero In-Out Deg Nodes:  41469
  Unique directed edges:     415174
  Unique undirected edges:   415027
  Self Edges:                26924
  BiDir Edges:               27218
  Closed triangles:          46992
  Open triangles:            69426319
  Frac. of closed triads:    0.000676
  Connected component size:  0.886745
  Strong conn. comp. size:   0.025758
  Approx. full diameter:     13
  90% effective diameter:    5.751723
```

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

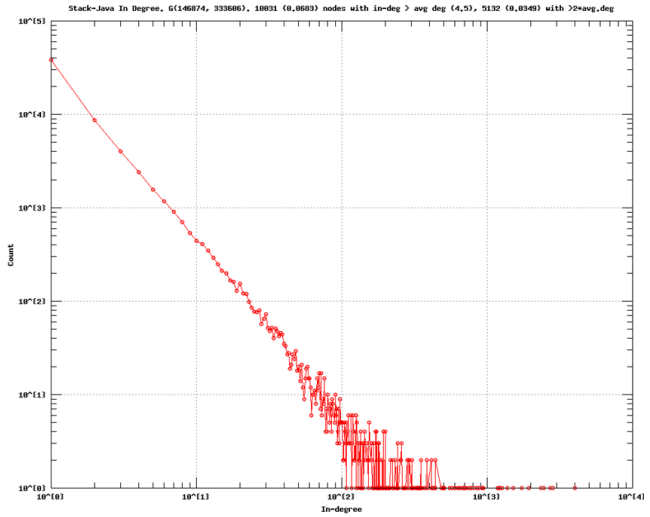
```
NId = snap.GetMxDegNId(GPA)
print "max degree node", NId

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

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
```

Get node with max degree

Get degree distribution



Node Centrality

- **Find "importance" of nodes in a graph**
  - PageRank, Hubs and Authorities (HITS)
  - 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

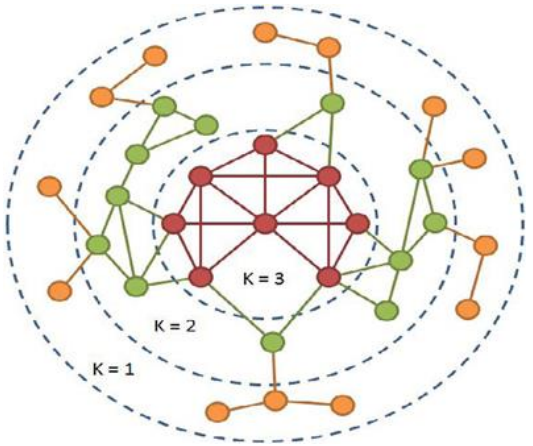
```
EigV = snap.TFltV()
snap.GetEigVec(G, EigV)

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

SNAP Network Datasets

Rok Sosič, Jure Leskovec
Stanford University

WWW-15, Florence, Italy **May, 2015**

SNAP Network Datasets

- http://snap.stanford.edu/data/
- **Public collection of large network datasets**
  - Over 15 network types
  - Over 70 datasets
  - Varying sizes from 20K up to 1.8B edges
- **Popular resource for network scientists**
  - Method development, study, benchmarking
- **Contribute your dataset**
  - We welcome new additions
- **SNAP Dataset Users mailing list**
  http://groups.google.com/group/snap-datasets

Datasets in SNAP (1)

- **Social networks**
  - Online social networks, edges represent interactions between users
- **Location-based online social networks**
  - Social networks with geographic check-ins
- **Online communities**
  - Data from online communities such as Reddit and Flickr
- **Networks with ground-truth communities**
  - Ground-truth network communities in social/information networks
- **Online reviews**
  - Data from online review systems such as Amazon
- **Amazon networks**
  - Nodes represent products, edges link co-purchased products
- **http://snap.stanford.edu/data/**

Datasets in SNAP (2)

- **Twitter and Memetracker**
  - Memetracker phrases, links and 467 million Tweets
- **Signed networks**
  - Networks with positive and negative edges (friend/foe, trust/distrust)
- **Communication networks**
  - Email communication networks with edges representing emails
- **Wikipedia networks and metadata**
  - Talk, editing and voting data from Wikipedia
- **Citation networks**
  - Nodes represent papers, edges represent citations
- **Collaboration networks**
  - Nodes represent scientists, edges represent collaboration (paper co-authoring)

- **http://snap.stanford.edu/data/**

Datasets in SNAP (3)

- **Web graphs**
  - Nodes represent webpages and edges are hyperlinks
- **Internet networks**
  - Nodes represent computers and edges communication
- **Autonomous systems**
  - Graphs of the internet
- **Road networks**
  - Nodes represent intersections and edges roads connecting the intersections
- **http://snap.stanford.edu/data/**

Social Circles from Facebook

- **Friends lists from Facebook**
  - Includes user profiles, circles, ego networks
  - Collected via Social Circles App on Facebook
    - **Contribute your own social circles:** http://snap.stanford.edu/socialcircles/
  - **Social circle detection Kaggle competition:**
    - https://www.kaggle.com/c/learning-social-circles

Dataset statistics	
Nodes	4039
Edges	88234
Nodes in largest WCC	4039 (1.000)
Edges in largest WCC	88234 (1.000)
Nodes in largest SCC	4039 (1.000)
Edges in largest SCC	88234 (1.000)
Average clustering coefficient	0.6055
Number of triangles	1612010
Fraction of closed triangles	0.2647
Diameter (longest shortest path)	8
90-percentile effective diameter	4.7

http://snap.stanford.edu/data/egonets-Facebook.html

Location Based Social Networks

Friendship network and check-ins in Gowalla location-based social network

Dataset statistics	
Nodes	196591
Edges	950327
Nodes in largest WCC	196591 (1.000)
Edges in largest WCC	950327 (1.000)
Nodes in largest SCC	196591 (1.000)
Edges in largest SCC	950327 (1.000)
Average clustering coefficient	0.2367
Number of triangles	2273138
Fraction of closed triangles	0.007952
Diameter (longest shortest path)	14
90-percentile effective diameter	5.7
Check-ins	6,442,890

http://snap.stanford.edu/data/loc-gowalla.html

Online Communities: Reddit

- **Post submissions to Reddit**
  - Includes an image with multiple submissions
  - **Features per posts:** number of ratings, the title, number of comments

Dataset statistics	
Number of submissions	132,308
Number of unique images	16,736
Average number of times an image is resubmitted	7.9
Timespan	July 2008 - Jan 2013

http://snap.stanford.edu/data/web-Reddit.html

Online Reviews: Amazon

- **18 years of Amazon reviews up to March 2013**
  - Product and user information, ratings, review text

Dataset statistics	
Number of reviews	34,686,770
Number of users	6,643,669
Number of products	2,441,053
Users with > 50 reviews	56,772
Median no. of words per review	82
Timespan	Jun 1995 - Mar 2013

http://snap.stanford.edu/data/web-Amazon.html

SNAP C++

Rok Sosič, Jure Leskovec
Stanford University

WWW-15, Florence, Italy **May, 2015**

SNAP C++ Installation

- **Download the latest version of SNAP C++**

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

Download SNAP

Current SNAP Release

Download the current SNAP distribution package:

SNAP 2.4 (May 11, 2015)

A public development SNAP repository is available at GitHub:

snap-stanford/snap

SNAP C++ Repository

- **Graph and network library:** directory **`snap-core`**
  - Graph and network generation, manipulation, algorithms
- **Data structures:** directory **`glib-core`**
  - STL-like library
  - Contains basic data structures, like vectors, hash tables and strings
  - Provides serialization for loading and saving
- **Tutorials:** directory **`tutorials`**
  - Short programs that demonstrate basic functionality
- **Example applications:** directory **`examples`**
  - Complete sample applications
- **Advanced capabilities:** directories **`snap-adv`**, **`snap-exp`**

SNAP Quick Start Guide

- **Download and unzip** **Snap package**
  - http://snap.stanford.edu/snap/download.html
- **Compile programs** in subfolder **`examples`**
  - Windows Visual Studio
    - `Project file` **`SnapExamples*.sln`**
  - Mac OS x with Xcode
    - `Project file` **`snap-examples*.xcodeproj`**
  - Command line on Linux, Mac OS X, Cygwin
    - **`Makefile`**
- For **your own project**, copy **`examples/testgraph`** and modify it

Installation on Windows

- Install Visual Studio or Visual Studio Express
  - http://www.visualstudio.com/
- **Download and Unzip Snap package**
  - http://snap.stanford.edu/snap/download.html
- Go to subfolder **`examples`**
- Open project **`SnapExamples*.sln`**
  - Visual Studio 2008 and 2010 projects are available

Visual Studio: Creating New Project

- **1)** Open Visual Studio and create a project
  - Or start with **`examples/testgraph`** and modify it
- **2)** Include **`Snap.h`** in your main program
  `#include "Snap.h"`
- **3)** Include the path to directories "**`snap-core`**", "**`glib-core`**" and "**`snap-adv`**" in your project
  - Properties → *Configuration Properties* → *VC++ Directories* → *Include Directories*
- **4)** Character set must be configured to Multi-Byte:
  - Properties → *Configuration Properties* → *General* → *Projects Defaults* → *Character Set* → *Select "Use Multi-Byte Character Set"*

Installation on Mac OS X with Xcode

- Install Xcode
  - https://developer.apple.com/xcode/
- **Download and Unzip Snap package**
  - http://snap.stanford.edu/snap/download.html
- Go to subfolder **`examples`**
- Open project **`snap-examples*.xcodeproj`**
- Build the project and execute examples

Xcode – Creating New Project

- **Open Xcode and create a project**
  - Or start with **`examples/testgraph`** and modify it

- Include "**`Snap.h`**" in your main program

```
#include "Snap.h"
```

Command Line Installation on Linux, Mac OS X, Windows with Cygwin

- For command line-based systems (e.g., Linux, OsX, Cygwin), use the **`Makefile`** in the example folder

- Makefiles are available in all folders in "**`examples`**", e.g., **`examples/kronfit/Makefile`**

Basic Graph Types

- **`TUNGraph`**: undirected graph
- **`TNGraph`**: directed graph
- **`TNEANet`**: directed multi-graph with attributes

Graph Creation

- **Create a graph:**

```
PNGraph Graph = TNGraph::New();
Graph->AddNode(1);
Graph->AddNode(5);
Graph->AddEdge(1,5);
```

- **Use smart-pointers**
  - `typedef TPt<TNGraph> PNGraph`
  - Memory management
    - Objects are automatically released when not needed

- Add nodes (`G->AddNode(i)`) before adding edges (`G->AddEdge(i,j)`)

Graph Traversal

- **Traverse the nodes**

```
for (TNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++)
  printf("%d %d %d\n",  NI.GetId(), NI.GetOutDeg(), NI.GetInDeg());
```

- **Traverse the edges, globally**

```
for (TNGraph::TEdgeI EI = Graph->BegEI(); EI < Graph->EndEI(); EI++)
  printf("edge (%d, %d)\n", EI.GetSrcNId(), EI.GetDstNId());
```

- **Traverse the edges, per node**

```
for (TNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++)
  for (int e = 0; e < NI.GetOutDeg(); e++)
    printf("edge (%d %d)\n", NI.GetId(), NI.GetOutNId(e));
```

Node and Edge Iterators

- **Get a node iterator from node id:**

```
TNGraph::TNodeI NI = Graph->GetNI(NId);
```

- **Get an edge iterator from node ids:**

```
TNGraph::TEdgeI EI = Graph->GetEI(SrcNId,DstNId);
```

Loading/Saving of Graphs

- **Loading a graph in the edge list, text format**

```
PUNGraph G2 =
      TSnap::LoadEdgeList<PUNGraph>("as20graph.txt",0,1);
```

  - 0, 1 are the columns of source, target nodes

- **Saving a graph in the edge list, text format**

```
TSnap::SaveEdgeList<PUNGraph>(G2, "as20graph.txt", "");
```

- **Loading/Saving in a binary format – faster**

```
{ TFIn FIn("test.graph");
  PNGraph G2 = TNGraph::Load(FIn); }
{ TFOut FOut("test.graph"); G2->Save(FOut); }
```

  - **Note the parenthesis {}!**

Edge List, Text File Format

- **Example file:**
  `as20graph.txt` in subfolder **`examples`**

```
# Directed Node Graph
# Autonomous systems …
# Nodes: 6474    Edges: 26467
# SrcNId   DstNId
1   3
1   6
1   32
1   48
1   63
1   70
…
```

Graph Operations (Examples 1)

- **Get degree distribution (degree, count)**

```
TSnap::GetOutDegCnt(G, CntV);
```

- **Get distribution of connected components (component size, count)**

```
TSnap::GetWccSzCnt(G, CntV);
```

  - CntV is a vector of pairs of integers:

```
TVec < TPair<TInt, TInt> > CntV;
```

Generating Graphs

- **Generate graphs with specific properties**
- Use functions **TSnap::Gen…**

  `TSnap::GenRndGnm()`: G_{nm} Erdős–Rényi graph

  `TSnap::GenForestFire`, Forest Fire Model

  `TSnap::GenPrefAttach`, Preferential Attachment

- **Example:**
  - Create a directed random graph on 100 nodes and 1k edges

```
PNGraph Graph =
TSnap::GenRndGnm<PNGraph>(100, 1000);
```

Graph Operations (Examples 2)

- **Generate a network using Forest Fire model**

```
PNGraph G = TSnap::GenForestFire(1000, 0.35, 0.35);
```

- **Convert to undirected graph TUNGraph**

```
PUNGraph UG = TSnap::ConvertGraph<PUNGraph, PNGraph> (G);
```

- **Get largest weakly connected component of G**

```
PNGraph WccG = TSnap::GetMxWcc(G);
```

- **Get a subgraph induced on nodes {0,1,2,3,4}**

```
PNGraph SubG = TSnap::GetSubGraph(G,
TIntV::GetV(0,1,2,3,4));
```

SNAP Network Types

- **`TNodeNet<TNodeData>`**: directed graph with TNodeData object for each node
- **`TNodeEDatNet<TNodeData, TEdgeData>`**: directed graph with TNodeData on each node and TEdgeData on each edge
- **`TNodeEdgeNet<TNodeData, TEdgeData>`**: directed multi-edge graph with TNodeData on each node and TEdgeData on each edge

Example Applications

- In SNAP directory "`examples`"
- **`TestGraph`**: Demonstrates basic functionality of the library, modify this example for your own project
- **`ForestFire`**: ForestFire graph generative model
- **`Cliques`**: Clique Percolation Method for detecting overlapping communities
- **`Cascades`**: Simulate susceptible-infected model on a network
- **`AGMFit, BigClam, CODA, Cesna`**: Community detection methods

SNAP Data Structures and Types

- In directory **`glib-core`**
- **Key files:**
  - **`dt.h`**: Data Types (`TInt, TFlt`)
  - **`ds.h`**: Data Structures (`TVec`)
- **Numbers:**
  - Integers: `TInt`
  - Real numbers: `TFlt`
  - Example:

```
TInt A = 5;
printf("%d\n", A.Val);
```

Basic SNAP Types

- **String: TStr**
  - **Examples:**

```
TStr A = "abc";
TStr B = "ccc";
printf("string %s\n", A.CStr());   // -- abc
printf("length %d\n", A.Len());    // -- 3
printf("A[0] %c\n", A[0]);         // -- a
printf("A==B %d\n", A == B);       // -- 0
```

SNAP Data Structures

- **Pair**
  - **TPair <Type1, Type2>**
    (Types can also be complex types like TVec, TPair…)

```
TPair<TInt, TFlt> A;
A.Val1 = 3;
A.Val2 = 3.14;
```

  - **Predefined types in ds.h**

```
typedef TPair<TInt, TInt> TIntPr;
typedef TPair<TInt, TIntPr> TIntIntPrPr;
```

- **Triple**
  - TTriple <Type1, Type2, Type3>

SNAP Vectors

- **TVec<Type>**
  - Example:

```
TVec<TInt> A;
A.Add(10);
A.Add(20);
A.Add(30);
printf("length %d\n", A.Len());    // -- 3
printf("A[0] %d\n", A[0].Val);     // -- 10
```

  - **"Type" can be a complex type**

```
TVec< TVec< TVec<TFlt> > >
```

  - **Predefined types in ds.h**

```
typedef TVec<TInt> TIntV;
Typedef TVec<TFlt> TFltV;
```

SNAP Hash Tables

- `THash <key, value>`
  - **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"

SNAP Hash Tables

- **Example:**

```
THash<TInt, TStr> A;
A.AddDat(100) = “David”;
A.AddDat(89) = “Ann”;
A.AddDat(95) = “Jason”;
printf(“%s\n”, A.GetDat(89).CStr()); // -- Ann, Key to Value
printf(“%d\n”, A.GetKeyId(95));      // -- 5, Key to KeyId
printf(“%d\n”, A.GetKey(5).Val);     // -- 95, KeyId to Key
printf(“%s\n”, A[5].CStr());         // -- Jason, KeyId to Value
```

- Predefined types in hash.h

```
typedef THash<TInt, TInt> TIntIntH;
Typedef THash<TInt, TFlt> TIntFltH;
```

Saving and Loading Objects

- **Binary files**
  - Fast save/load
  - Memory efficient
- **Save():**

```
TIntStrH A;
{ TFOut fout("a.bin");
  A.Save(fout); }
```

- **Load():**

```
{ TFIn fin("a.bin"); A.Load(fin); }
```

Generating Distributions

- **TRnd class**
  - Generate random numbers according to various probability distributions
- **Example:**

```
TRnd A;
//sample from an exponential distribution
for (int i=0; i<10; ++i){
  printf("%f\n",A.GetExpDev(1));
}
```

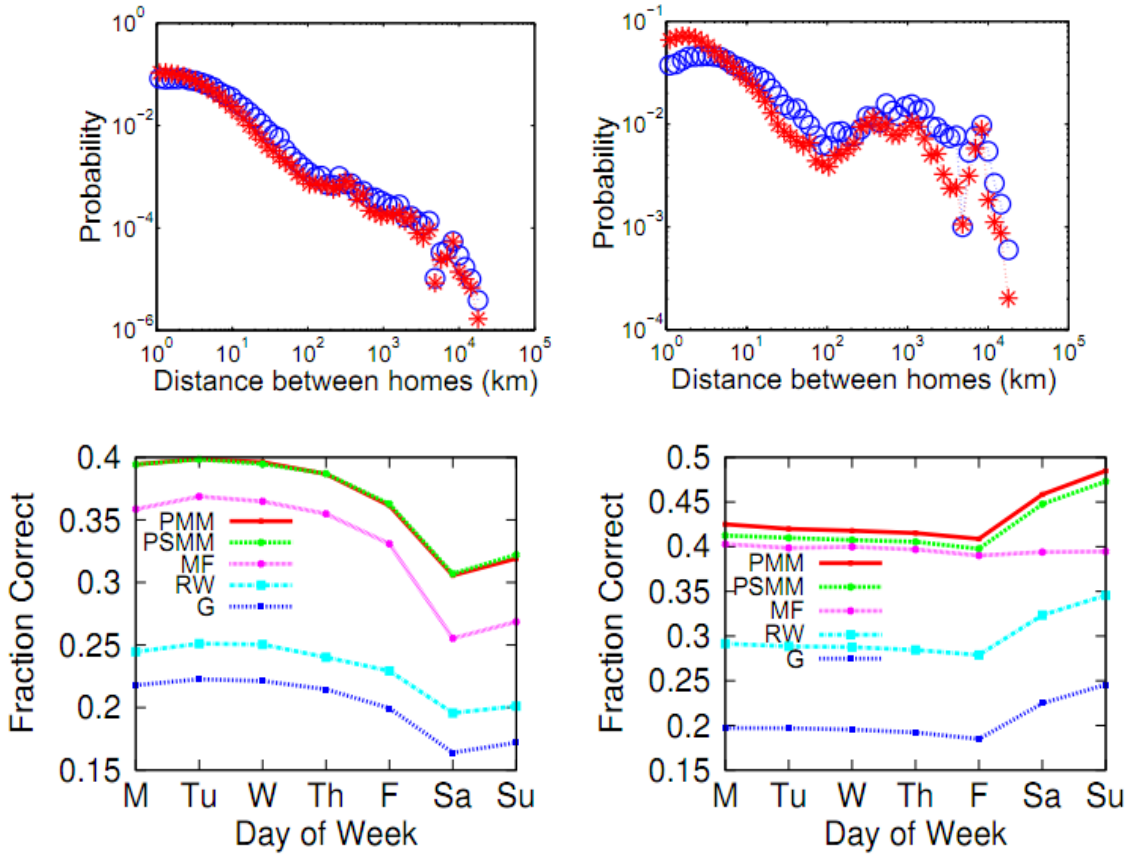
Calculating Statistics

- **File `glib-core/xmath.h`**
  - Useful for calculating moments, correlation coefficients, t-test, …
- **Example of computing moments (`TMom`):**

```
TMom Mom;
Mom.Add(5);  Mom.Add(6);  Mom.Add(8);
Mom.Def();
printf("Avg: %f\n", Mom.GetMean());
printf("Min: %f\n", Mom.GetMn());
printf("Max: %f\n", Mom.GetMx());
```

Making Plots

- **Making a plot in SNAP**



Making Plots in SNAP

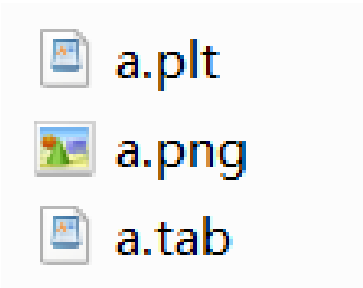
- **1) Install Gnuplot** http://www.gnuplot.info/
  - Make sure that the directory containing wgnuplot.exe (for Windows) or gnuplot (for Linux, Mac OS X) is in your environmental variable $PATH.

- **2)** Use **`TGnuPlot`** (`glib-core/gnuplot.h`):

```
TVec<TPair<TFlt, TFlt > > XY1, XY2; …
TGnuPlot Gp(“file name”, “title name");
Gp.AddPlot(XY1, gpwLinesPoints, “curve1");
Gp.AddPlot(XY2, gpwPoints, “curve2");
Gp.SetXYLabel(“x-axis name", “y-axis name”);
Gp.SavePng(); //or Gp.SaveEps();
```

Gnuplot in SNAP

- **After executing, three files are generated**



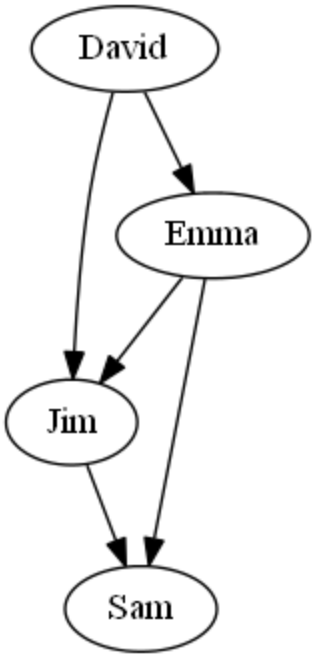
  - **.plt** file includes plotting commands for gnuplot
  - **.tab** file contains the tab separated data
  - **.png** or **.eps** is the plot

Drawing SNAP Graphs

- Use **`TGraphViz`**
  - Need to install GraphViz software first http://www.graphviz.org/
- Add GraphViz path to environment variable

Drawing SNAP Graphs

```
PNGraph G = 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);
TIntStrH Name;
Name.AddDat(1)="David";
Name.AddDat(2)="Emma";
Name.AddDat(3)="Jim";
Name.AddDat(4)="Sam";
TGraphViz::Plot<PNGraph>(G, gvlDot,
      "gviz_plot.png", "", Name);
```



SNAP Hands-on Exercise

Rok Sosič, Jure Leskovec
Stanford University

WWW-15, Florence, Italy **May, 2015**

Stack Overflow Dataset

- Publicly available by Stack Overflow

  https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z

- 6.6GB compressed, 33GB uncompressed

- From Jul 2008 to Apr 2015
  - 8,978,719 questions, 15,074,572 answers

Hands-on Exercise

- **Task:**
  - Find top Java experts on Stack Overflow
- **Possible approaches for finding experts:**
  - Use Stack Overflow reputation score:
    - Not Java specific
    - No control
  - Count the number of answers:
    - No measure of answer importance or usefulness
  - Create a social network and compute **user centrality**:
    - PageRank, HITS

Finding Top Java Experts

- **Plan:**
  - Use node centrality measure, PageRank
  - Need a graph
- **Constructing a graph:**
  - Nodes, each user a node
  - Edges, a question owner points to the owner of the accepted answer

Finding Top Java Experts

- **Method Overview:**
  - Step 1: Extract relevant fields from input
  - Step 2: Select questions about Java
  - Step 3: Build the graph
    - Find owners of accepted answers
  - Step 4: Analyze the graph

Stack Overflow: Questions

- **Questions XML format in Posts.xml:**
  - Total 8,978,719 questions, Java 810,071

```
<row Id="4“ PostTypeId="1“
 OwnerUserId="8“ AcceptedAnswerId="7“
 Tags="&lt;c#&gt;&lt;winforms&gt;&lt;forms&gt;
   &lt;opacity&gt;" .. />
```

Field	Value
Id	4
PostTypeId	1
OwnerUserId	8
Accepted AnswerId	7
Tags	c#, winforms, forms, opacity

Stack Overflow: Answers

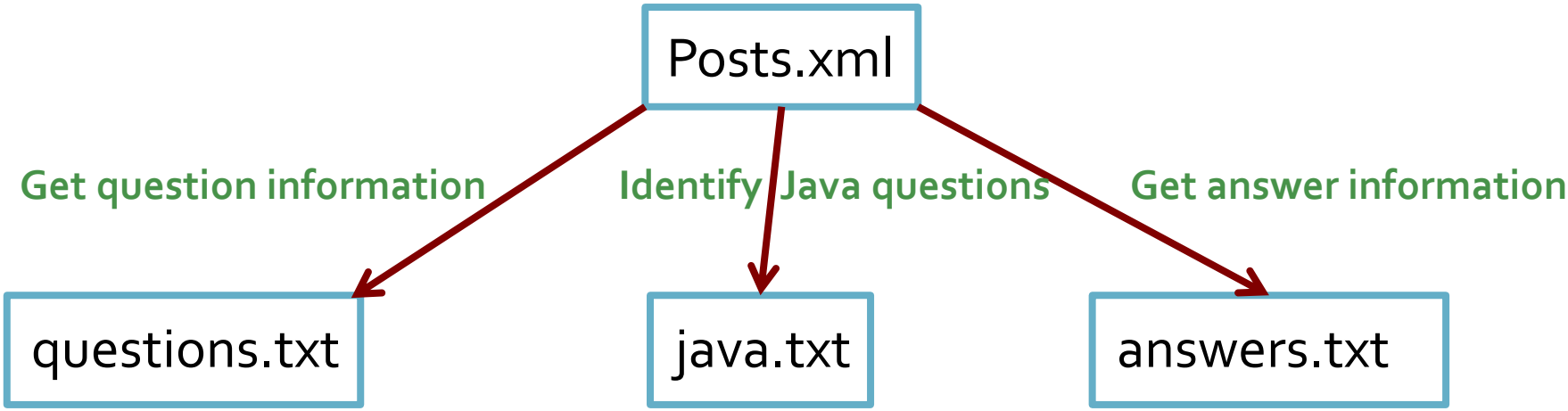
- **Answers XML format in Posts.xml:**
  - Total 15,074,572

```
<row Id="12" PostTypeId="2" OwnerUserId="1" ... />
```

Field	Value
Id	12
Post TypeId	2
OwnerUserId	1

Workflow to Find Java Experts

- **Step 1, Process input file, extract relevant fields**
  - Get lists of questions and answers, identify Java posts
  - Convert XML format to TSV (tab separated values)



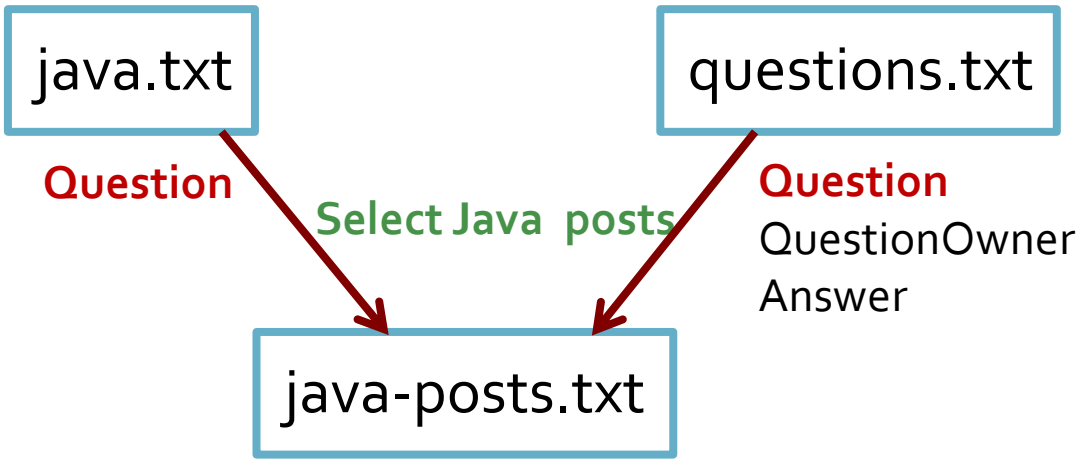
getQuestions.py Posts.xml
Question
QuestionOwner
Answer

getTag.py Posts.xml java
Question

getAnswers.py Posts.xml
Answer
AnswerOwner

Workflow to Find Java Experts

- **Step 2**, Select only Java related questions

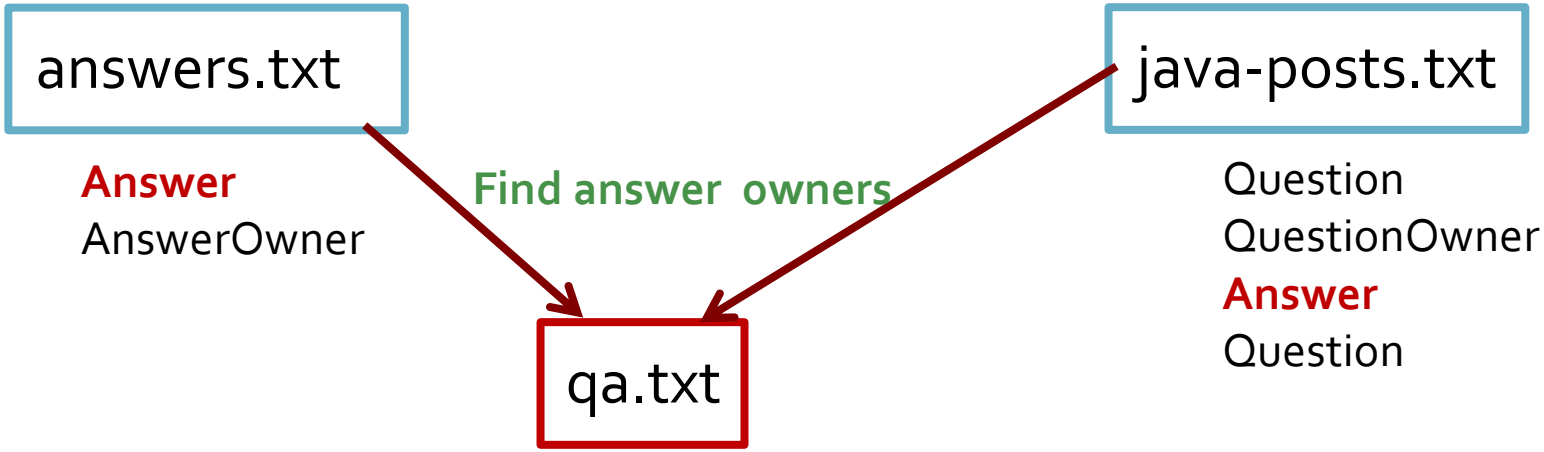


doJoin.py java.txt questions.txt 1 1

Question
QuestionOwner
Answer
Question

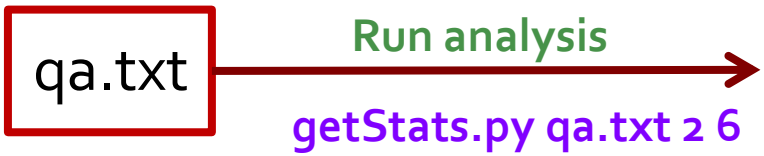
Workflow to Find Java Experts

- **Step 3, Build the graph by finding owners of accepted answers**



Workflow to Find Java Experts

- **Step 4, Analyze the graph**
  - Find top Java experts



Question
QuestionOwner
Answer
Question
Answer
AnswerOwner

- Program calculations
  - # of nodes, edges
  - Distribution of weakly connected components
  - In and out-degree distributions
  - Top 10 experts by PageRank
  - Top 10 experts by HITS
  - Top 10 learners by HITS

```
top 10 experts by PageRank
id   22656, pagerank 0.007056
id  139985, pagerank 0.005290
id  571407, pagerank 0.004348
id  992484, pagerank 0.003722
id  157882, pagerank 0.003628
...
```

Java Experts Graph

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

Output:

```
QA Stats: Directed
  Nodes:                     188406
  Edges:                     415174
  Zero Deg Nodes:            0
  Zero InDeg Nodes:          108618
  Zero OutDeg Nodes:         38319
  NonZero In-Out Deg Nodes:  41469
  Unique directed edges:     415174
  Unique undirected edges:   415027
  Self Edges:                26924
  BiDir Edges:               27218
  Closed triangles:          46992
  Open triangles:            69426319
  Frac. of closed triads:    0.000676
  Connected component size:  0.886745
  Strong conn. comp. size:   0.025758
  Approx. full diameter:     13
  90% effective diameter:    5.751723
```

Java Experts on Stack Overflow

- **Comparing methods on top 10 results:**

http://stackoverflow.com/users/<id>

In-degree	RageRank	HITS
22656	22656	22656
571407	139985	571407
992484	571407	57695
157882	992484	139985
57695	157882	157882
139985	57695	203907
522444	218978	992484
131872	70604	522444
438154	230513	131872
207421	438154	438154

Java Learners on Stack Overflow

- **Comparing methods on top 10 results:**

  http://stackoverflow.com/users/<id>

Out-degree	HITS
1194415	892029
892029	1194415
785349	359862
470184	648138
454049	470184
853836	802050
359862	384706
44330	225899
663148	454049
1379286	130758

Finding Top Java Experts

- **Solution:**
  - Step 1: Extract relevant fields from input

```
python getQuestions.py Posts.xml > questions.txt
python getAnswers.py Posts.xml > answers.txt
python getTag.py Posts.xml java > java.txt
```

  - Step 2: Select questions about Java

```
python doJoin.py java.txt questions.txt 1 1 > \
                java-posts.txt
```

  - Step 3: Build the graph

```
python doJoin.py answers.txt java-posts.txt 1 3 > \
                qa.txt
```

  - Step 4: Analyze the graph

```
python getStats.py qa.txt 2 6 > stats.txt
```

Find Java Experts: Hands-on Exercise

- **Download and install Snap.py**
  http://snap.stanford.edu/snappy/index.html
- **Download programs and data for the exercise: www15-code.zip** and **www15-data.zip**, for finding experts on Stack Overflow
  http://snap.stanford.edu/proj/snap-icwsm
- **Unpack** zip files www15-code.zip and www15-data.zip
- **Find experts** by executing the following programs from command line
  - **stackoverflow.sh** on Mac OS X and Linux
  - **stack.bat** on Windows
  - **stats.txt** contains the output
- **Explore getStats.py**
  - Extend it with different graph analysis methods
- **Extra exercise**
  - Find Javascript experts, change in experts over time
- Stack Overflow original data - 6.6GB!
  https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z

Contact information: Rok Sosič, rok@cs.stanford.edu