Introduction to Snap.py

CHEONG YIUFUNG

ZHENG RUI

COMP4641 Labs

TAs:

- Zheng Rui (rzhengac)
- Zhang Yaofeng or Cheong Yiufung (yzhangak)

Labs:

- SNAP(Stanford Network Analysis Platform)
- NetworkX
- Graph Theory
- Probability
- Data Mining Techniques
- Crawler
- Homeworks and projects
- More? Tell us what you want to learn

SNAP

- **S**tanford **N**etwork **A**nalysis **P**roject (SNAP)
- A general purpose library for network analysis and graph mining
- Originally written in C++
- You can use SNAP to construct and manipulate large graphs, calculates structural properties, generate graphs from built-in models
- Useful for your projects!
- More info on http://snap.stanford.edu

Snap.py

- SNAP for Python
  - Provides SNAP functionality in Python
- C++: fast execution, yet difficult to write and requires compilation
- Python: simple language, interactive use
- Snap.py: fast execution in a simple language, interactive use
- More info at: http://snap.stanford.edu/snappy/index.html

Content

- Installation
- Tutorial
- Q&A

Content

- **Installation**
- Tutorial
- Q&A

Installation

- Download and install via: http://snap.stanford.edu/snappy/index.html#download
- Since we're in lab, we will talk about installation under Windows.
  - Linux and OS X users please follow instructions on the website
- Requirements for Windows:
  - 64-bit OS, 64-bit python, Visual C++ Redistributable for Visual Studio 2012
  - For your convenience, a package available at course website, containing python, VS redistributable and latest snap.py.

Installation steps

1. Install python-amd64.msi
2. Install VS redistributable
3. Unzip latest snap.py
4. In cmd, run “python setup.py install” under unzipped folder
5. Run “python quick_test.py” to verify installation

Content

- Installation
- **Tutorial**
- 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

Starter

We will use IDLE, the Python GUI for illustration.

- You can also run python under cmd.

Make sure you import **snap** module first:

- For convenience, we will just import everything

Python 2.7.9 Shell

File Edit Shell Debug Options Windows Help

```
Python 2.7.9 (default, Dec 10 2014, 12:28:03) [MSC v.1500 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> from snap import *
>>>
```

Basic Types: T<type_name>

- **TInt**: int
- **TFlt**: float
- **TStr**: str (Don’t use empty str “” as argument)
- Automatically converted between C++ and Python
  - Normally no need to deal with basic types explicitly

Vector Types: T<type_name>V

- Vectors: sequences of values of the same type
  - TIntV: a vector of integer type. Same for TStrV and TFltV

- Operations:
  - Add(): add element
  - Len: get size
  - [index]:get value
  - SetVal(index, new_val):modify value
  - for i in v: iterator

```
>>> v = TIntV() # Creates an empty integer vector
>>> v.Add(1) # Add elements
0
>>> v.Add(2)
1
>>> v.Add(3)
2
>>> v.Len() # Get vector length
3
>>> v[2] # Retrieve value with index
3
>>> v.SetVal(2,7) # Change value with index
>>> v[2]
7
>>> for item in v: # print values in a vector
        print item



1
2
7
```

Hash Table Type: T<key_type><val_type>H

- Hash table:
  - When key and val is of the same type, just use once, e.g TIntH: Int -> Int
  - Stores <key, value> pairs, where keys and values must be of the same type respectively

- Operations:
  - []: add a new or change an existing value, get value
  - Len: size
  - for i in h: iterator

```
>>> h = TIntStrH()
>>> h[1] = "January"
>>> h[5] = "May"
>>> h[2] = "February"
>>> h.Len()
3
>>> h[5]
'May'
>>> h[2] = "Wake me up when the semester ends"
>>> h[2]
'Wake me up when the semester ends'
>>> for k in h:
        print k, h[k]


1 January
5 May
2 Wake me up when the semester ends
>>>
```

Pair Types T<type1><type2>Pr

- Pair contains two values.
- Different from Hash, the two values can be of different types.

- Operation:
  - GetVal1(): get first value
  - GetVal2(): get second value

```
>>> pr = TIntStrPr(1, "one")
>>> pr.GetVal1()
1
>>> pr.GetVal2()
'one'
```

Graphs and Networks

- Graphs: describe only topologies
  - Nodes with unique integer ids
- Networks: Graphs with data on nodes and/or edges of the network
- Graph classes:
  - TUNGraph: undirected graph
  - TNGraph: directed graph
- Network class:
  - TNEANet: directed graphs with attributes for nodes and edges

Graphs and Networks

- Pointers to graphs, names start with **P**
  - **PUNGraph, PNGraph, PNEANet**
  - Class methods (functions) use **T**
  - Instances (variables) use **P**

```
>>> G1 = TUNGraph.New() # Graph creation
>>> G1.AddNode(1)
1
>>> G1.AddNode(5)
5
>>> G1.AddEdge(1, 5) # Add nodes before edges
-1
```

```
...
>>> G2 = GenRndGnm(PNGraph, 100, 1000) # 100 nodes with 1000 edges
>>> for EI in G2.Edges():
        print "edge: (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())


edge: (0, 5)
edge: (0, 7)
edge: (0, 12)
edge: (0, 16)
edge: (0, 19)
edge: (0, 34)
edge: (0, 36)
edge: (0, 41)
edge: (0, 43)
edge: (0, 49)
edge: (0, 56)
edge: (0, 86)
edge: (1, 0)
edge: (1, 24)
```

Graph Traversal

For Nodes:

- GetId(): get integer id
- GetOutDeg(): out degree
- GetInDeg(): in degree

For Edges:

- GetSrcNId(): get source node id
- GetDstNId(): get dest node 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)
```

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)
```

Graph Manipulations

```
# 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

```
# define a vector of pairs of integers (size, count) and
# get a distribution of connected components (component size, count)
G9 = snap.GenRndGnm(snap.PNGraph, 10000, 1000)
CntV = snap.TIntPrV()
snap.GetWccSzCnt(G9, CntV)
for p in CntV:
print "size %d: count %d" % (p.GetVal1(), p.GetVal2())
```

Plotting

- You can choose from:
  - Gnuplot: http://www.gnuplot.info/
  - Graphviz: http://www.graphviz.org/
  - Matplotlib: http://matplotlib.org/

- Follow instructions on respective websites.

Datasets

- 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

Useful Link

- Snap.py Reference Manual: http://snap.stanford.edu/snappy/doc/reference/index-ref.html
  - All functions with detailed documentation

- Check also Snap in C++: http://snap.stanford.edu/snap/index.html

Content

- Installation
- Tutorial
- **Q&A**