2015 Fall Term: Web Analytics IS 620

Week Three - Network Analysis: Graph Theory, Definitions

Author: Partha Banerjee

In [1]:

```
%matplotlib inline

import Tkinter, tkFileDialog, csv, sys
import networkx as nx
import matplotlib.pyplot as plt
g=nx.Graph()
```

1. Load a graph database of your choosing from a text file or other source. If you take a large network dataset from the web (such as from https://snap.stanford.edu/data/ (https://snap.stanford.edu/data/)), please feel free at this point to load just a small subset of the nodes and edges.

In [2]:

```
try:
    # Read datafile name with path
    input_file = tkFileDialog.askopenfilename()

    # This input file has downloaded from https://snap.stanford.edu/data/ca
    # Directed graph (each unordered pair of nodes is saved once): CA-GrQc.
    # Collaboration network of Arxiv General Relativity category (there is
    # Nodes: 5242 Edges: 28980

    # Modified the datafile (by deleting records to reduce the execution ti
    #input_file = "C:\\Partho\\MSDA\\Dropbox\\IS 620 Web Analytics\Wk 3 - N

    # Read file
    fromNode = []
    toNode = []
    with open(input_file, "r") as fl:
        allrecs = csv.reader(fl)
        next(allrecs, None)            # Skip header
        try:
            count = 0
            for row in allrecs:
                # Keep it small, take only 100 records
                if (count==100):
                    break
                if row[0] not in fromNode:
                    fromNode.append(row[0])
                    g.add_node(row[0])
                if row[1] not in toNode:
                    toNode.append(row[1])
                    g.add_node(row[1])
                g.add_edge(row[0],row[1])
                count += 1
        except csv.Error as er:
            sys.exit('File %s, line %d: %s' % (input_file, allrecs.line_num
            exit(-1)
        except IndexError:
            print "Data Error, exiting..."
            exit(1)

    # Take only unique data values
    fromNode = set(fromNode)
    toNode = set(toNode)

    N = len(fromNode)
    if N > 0:
        print "Total fromNode in datafile %s: %d" % (input_file, len(fromNo
        print "Total toNode in datafile  %s: %d" % (input_file, len(toNode)
        print
    else:
        print "Datafile has no fromNode data in it, Exiting..."
except IOError:
    # User presses Cancel button instead of selecting a file
    print "Sorry, you have aborted File selection option!!!"
```

```
Total fromNode in datafile C:\Partho\MSDA\Dropbox\IS 620 Web Analytics\Wk
3 - Network Analysis Graph Theory, Definitions\CA-GrQc-mod.csv: 5
Total toNode in datafile  C:\Partho\MSDA\Dropbox\IS 620 Web Analytics\Wk
3 - Network Analysis Graph Theory, Definitions\CA-GrQc-mod.csv: 15
```

In [3]:

```
fromNode
```

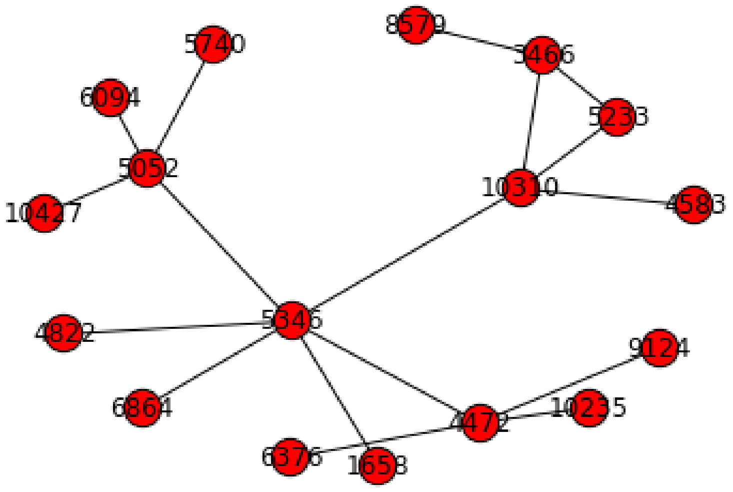
Out[3]:

```
{'10310', '3466', '4472', '5052', '5346'}
```

2. Create basic analysis on the graph, including the graph's diameter, and at least one other metric of your choosing. You may either code the functions by hand (to build your intuition and insight), or use functions in an existing package.

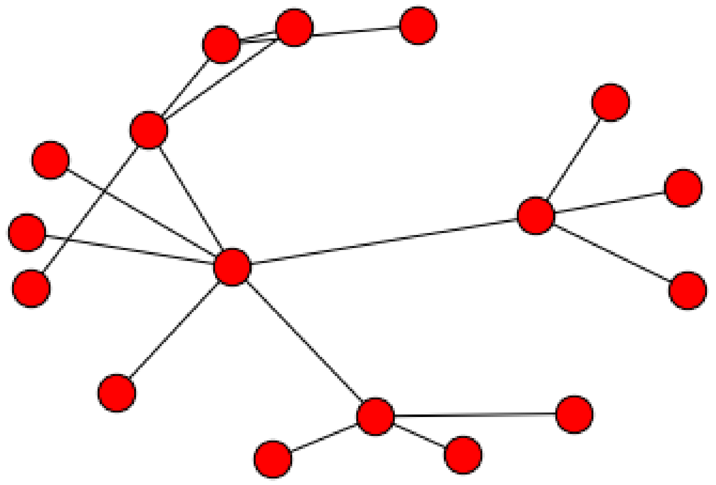
In [4]:

```
pos=nx.spring_layout(g)
nx.draw(g,pos)
labels=nx.draw_networkx_labels(g,pos)
plt.show()
```



Readability with label is bad, unable to find ways to resize the graph height/size. So putting the same graph without label below.

In [5]:
```
pos=nx.spring_layout(g)
nx.draw(g,pos)
plt.show()
```



In [6]:
```
# Now let us check few matrices - starting with the DFS
from networkx import algorithms
from networkx.algorithms import traversal
edges = traversal.dfs_edges(g)
list(edges)
```

Out[6]:
```
[('3466', '10310'),
 ('10310', '5346'),
 ('5346', '4822'),
 ('5346', '1658'),
 ('5346', '6864'),
 ('5346', '4472'),
 ('4472', '9124'),
 ('4472', '10235'),
 ('4472', '6376'),
 ('5346', '5052'),
 ('5052', '5740'),
 ('5052', '6094'),
 ('5052', '10427'),
 ('10310', '5233'),
 ('10310', '4583'),
 ('3466', '8579')]
```

In [7]:
```
traversal.dfs_successors(g)
```

Out[7]:
```
{'10310': ['5346', '5233', '4583'],
 '3466': ['10310', '8579'],
 '4472': ['9124', '10235', '6376'],
 '5052': ['5740', '6094', '10427'],
 '5346': ['4822', '1658', '6864', '4472', '5052']}
```

In [8]:
```
tree = traversal.dfs_tree(g, '10310')
tree.succ
```

Out[8]:
```
{'10235': {},
 '10310': {'3466': {}, '4583': {}, '5346': {}},
 '10427': {},
 '1658': {},
 '3466': {'5233': {}, '8579': {}},
 '4472': {'10235': {}, '6376': {}, '9124': {}},
 '4583': {},
 '4822': {},
 '5052': {'10427': {}, '5740': {}, '6094': {}},
 '5233': {},
 '5346': {'1658': {}, '4472': {}, '4822': {}, '5052': {}, '6864': {}},
 '5740': {},
 '6094': {},
 '6376': {},
 '6864': {},
 '8579': {},
 '9124': {}}
```

In [9]:
```
edges = traversal.bfs_edges(g, '10310')
list(edges)
```

Out[9]:
```
[('10310', '3466'),
 ('10310', '5346'),
 ('10310', '5233'),
 ('10310', '4583'),
 ('3466', '8579'),
 ('5346', '4822'),
 ('5346', '1658'),
 ('5346', '6864'),
 ('5346', '4472'),
 ('5346', '5052'),
 ('4472', '9124'),
 ('4472', '10235'),
 ('4472', '6376'),
 ('5052', '5740'),
 ('5052', '6094'),
 ('5052', '10427')]
```

```
In [10]: traversal.bfs_successors(g, '10310')

Out[10]: {'10310': ['3466', '5346', '5233', '4583'],
          '3466': ['8579'],
          '4472': ['9124', '10235', '6376'],
          '5052': ['5740', '6094', '10427'],
          '5346': ['4822', '1658', '6864', '4472', '5052']}
```

```
In [11]: traversal.dfs_successors(g, '10310')

Out[11]: {'10310': ['3466', '5346', '4583'],
          '3466': ['5233', '8579'],
          '4472': ['9124', '10235', '6376'],
          '5052': ['5740', '6094', '10427'],
          '5346': ['4822', '1658', '6864', '4472', '5052']}
```

```
In [12]: algorithms.shortest_path(g,'10310','4472')

Out[12]: ['10310', '5346', '4472']
```

```
In [13]: algorithms.average_shortest_path_length(g)

Out[13]: 2.823529411764706
```

End of Homework, #3 is due for presentation on Thu 9/17 meetup.