```python
# Tony Hyun Kim
# 2013 10 03
# CS 224w, PS 1, Problem 2

import random
import snap

numNodes = 5242
numEdges = 14496

# Purely random (Erdos-Renyi) graph
G1 = snap.GenRndGnm(snap.PUNGraph, numNodes, numEdges)

# Small-world network
G2 = snap.TUNGraph.New()

for nid in range(numNodes):
    G2.AddNode(nid)

for nid in range(numNodes):
    for x in [-2, -1, 1, 2]:
        G2.AddEdge(nid, (nid+x)%numNodes)

numRndEdges = 4012
x = 0 # Counter for number of random edges added
while (x < numRndEdges):
    n1 = random.randint(0, numNodes-1)
    n2 = random.randint(0, numNodes-1)
    if (not G2.IsEdge(n1,n2) and (n1 != n2)): # Optionally block self-loops
        G2.AddEdge(n1,n2)
        x += 1

# Real-world collaboration graph
source = 'ca-GrQc.txt'
G3 = snap.LoadEdgeList(snap.PUNGraph, source, 0, 1)

# Part a: Get the degree distribution
#   Surely, Snap.py has a mechanism for exporting data into a file.
#   In this assignment, I just piped the console output into a file.
#-------------------------------------------------------------
G = G2
deg_hist = snap.TIntPrV()
snap.GetDegCnt(G, deg_hist)
'''
for item in deg_hist:
    print "{} {}".format(item.GetVal1(), item.GetVal2())
'''

# Part b: Compute the excess distribution
#   Below, I compute the excess degree distribution by the explicit
#   formula provided in the assignment, even though I was able to
#   derive a closed-form formula for qk in terms of pk
#-------------------------------------------------------------
G = G2

# Roundabout way to get the max degree of a graph...
max_degree = (G.GetNI(snap.GetMxDegNId(G))).GetDeg()
excess_deg_hist = [0]*max_degree

# Literal interpretation of the provided definition
'''
for k in range(max_degree): # Fix k
    for ni in G.Nodes():
        for j in ni.GetOutEdges():
            if (G.GetNI(j).GetDeg() == k+1):
                excess_deg_hist[k] += 1
'''

# Slightly more efficient calculation
for ni in G.Nodes():
    for j in ni.GetOutEdges():
        excess_deg_hist[G.GetNI(j).GetDeg()-1] += 1
```

```
'''
for k in range(max_degree):
    if (excess_deg_hist[k] > 0):
        print "{} {}".format(k, excess_deg_hist[k])
'''

# Part c: Compute the clustering coefficient "manually"
#-------------------------------------------------------------
G = G2
for ni in G.Nodes():
    ni_neighbors = snap.TIntV()
    ni_neighbors.Add(ni.GetId())
    ki = ni.GetOutDeg()
    if (ki < 2): # Ignore nodes with 0 or 1 neighbor
        break
    for nj_id in ni.GetOutEdges():
        ni_neighbors.Add(nj_id)
    G_ind = snap.GetSubGraph(G, ni_neighbors)
    ei = G_ind.GetEdges() - ki
    print "{} {} {}".format(ni.GetId(), ki, ei)
```