```
# Tony Hyun Kim
# CS 224w, PS 4, Problem 2
#-----------------------------------------------------------------
from __future__ import division
import random
import snap

# Generate the random graph according to Barabasi-Albert PA model
#-----------------------------------------------------------------
n = 1000
G = snap.GenPrefAttach(n, 4, snap.TRnd()) # Undirected graph

# I will use NG to store the betweenness centrality
# Note that the PNEANet type stores directed edges. Since the
#   actual graph of interest (G) is undirected, we will access
#   data in NG where the nodes of an edge are in increasing order
NG = snap.ConvertGraph(snap.PNEANet, G)
NG.AddFltAttrE("DeltaE", 0.0)
NG.AddIntAttrE("k", 0)

# Parameters for Algorithm 2
c = 5
max_iter = n//10

# Compute the approximate betweenness centrality
#-----------------------------------------------------------------
sids = random.sample(xrange(n), max_iter)
for sid in sids:
    s = G.GetNI(sid)

    # Obtain the BFS tree as a network, so that we can use attributes
    Ts = snap.ConvertGraph(snap.PNEANet,
                           snap.GetBfsTree(G, sid, True, False))
    Ts.AddIntAttrN("sigma", 0)   # Number of shortest paths
    Ts.AddFltAttrE("delta", 0.0) # Dependency

    # Girvan-Newman algorithm requires us to walk down BFS tree for sigma
    #   climb back up for delta
    fwd_order = [x.GetId() for x in Ts.Nodes()]
    fwd_order.pop(0) # Don't need to go over the source node
    rev_order = fwd_order[::-1]

    # Perform downward BFS pass on sigma
    Ts.AddIntAttrDatN(sid, 1, "sigma")
    for vid in fwd_order:
        v = Ts.GetNI(vid)
        sigma = 0
        for parent_id in v.GetInEdges():
            sigma += Ts.GetIntAttrDatN(parent_id, "sigma")
        Ts.AddIntAttrDatN(vid, sigma, "sigma")

    # Perform upward BFS pass on delta
    for wid in rev_order:
        w = Ts.GetNI(wid)

        multiplier = 1
        for child_id in w.GetOutEdges():
            multiplier += Ts.GetFltAttrDatE(Ts.GetEId(wid, child_id), "delta")

        for vid in w.GetInEdges():
            delta  = Ts.GetIntAttrDatN(vid,"sigma")/Ts.GetIntAttrDatN(wid,"sigma")
            delta *= multiplier
            Ts.AddFltAttrDatE(Ts.GetEId(vid, wid), delta, "delta")

    # Accumulate deltas
    for ei in G.Edges():
        e = sorted([ei.GetSrcNId(), ei.GetDstNId()])
        e_bc = NG.GetEId(e[0], e[1])
        De   = NG.GetFltAttrDatE(e_bc, "DeltaE")
        if (De < c*n): # Accumulate only when De < c*n
            k = NG.GetIntAttrDatE(e_bc, "k")
            NG.AddIntAttrDatE(e_bc, k+1, "k")
```

```
            e_ts = Ts.GetEId(e[0], e[1])
            if (e_ts < 0): # The tree Ts is directed
                e_ts = Ts.GetEId(e[1], e[0])
            if (e_ts < 0): # It may be the case that the edge does not occur in Ts
                continue
            De += Ts.GetFltAttrDatE(e_ts, "delta")
            NG.AddFltAttrDatE(e_bc, De, "DeltaE")


# Display the final betweenness centrality
for ei in G.Edges():
    e = sorted([ei.GetSrcNId(), ei.GetDstNId()])
    e_bc = NG.GetEId(e[0], e[1])
    De   = NG.GetFltAttrDatE(e_bc, "DeltaE")
    k    = NG.GetIntAttrDatE(e_bc, "k")
    print "{} {} {:.4f} {}".format(e[0], e[1], De, k)
```