

```

# Tony Hyun Kim
# CS 224w, PS 4, Problem 4.7
# Implementation of TwoStepGreedy
#-----
from __future__ import division

import snap
import sys

# Some support functions
#-----
def GetGraphCopy(G):
    G2 = snap.TUNGraph.New()
    for n in G.Nodes():
        G2.AddNode(n.GetId())
    for e in G.Edges():
        G2.AddEdge(e.GetSrcNid(), e.GetDstNid())
    return G2

def RemoveCore(G, C2, A, S):
    """
    G (snap.TUNGraph)
    C2: List of node IDs in the unanchored 2-core
    A: List of node IDs of the anchored set
    S: List of node IDs of the saved set
    """
    N = []
    N.extend(C2)
    N.extend(A)
    N.extend(S)

    for u in N:
        for v in N:
            if (u != v):
                G.DelEdge(u,v)

def FindLongestBranchInTree(GT, root_id):
    """
    GT (snap.TNGraph): Directed tree
    root_id (int): ID of the root node in tree

    Returns:
    branch: List of node ids in the longest branch
    """
    # First, compute the maximum hop in the tree
    HopCntV = snap.TIntPrV()
    snap.GetNodesAtHops(GT, root_id, HopCntV, True)

    max_hop = -1
    for HopCnt in HopCntV:
        hop = HopCnt.GetVall()
        if (hop > max_hop):
            max_hop = hop

    # Second, find the leaf node in the longest branch
    leaves = snap.TIntV()
    snap.GetNodesAtHop(GT, root_id, max_hop, leaves, True)

    # Finally, enumerate the branch in reverse order, i.e.
    # from the leaf back to the root. Note that for k=2 core
    # the nodes in the BFS tree cannot have more than one
    # parent (otherwise we have a contradiction)
    branch = []

    nid = leaves[0]
    while (nid != root_id):
        branch.append(nid)
        n = GT.GetNI(nid)
        nid = n.GetInEdges().next()

    return branch

# Load the graph

```

```

#-----
#source = "g1.txt"
source = sys.argv[1]
G = snap.LoadEdgeList(snap.PUNGraph, source, 0, 1)
#print "Loaded graph '{}'.format(source)

# Precompute the 2-core of G
G_C2 = snap.GetKCore(G, 2)
C2 = []
for n in G_C2.Nodes():
    C2.append(n.GetId())
#print "The 2-core of unanchored graph has {} nodes".format(len(C2))

# Maximum budget to consider
b_max = int(sys.argv[2])
b      = b_max

A = [] # List of anchored nodes
S = [] # List of nodes saved by anchored nodes

# Implementation of TwoStepGreedy!
B1 = [] # Best (longest) branch
B2 = [] # Second-best branch

while b > 0:
    #print "* Remaining budget: b={}".format(b)
    G2 = GetGraphCopy(G)
    RemoveCore(G2, C2, A, S)

    # Loop over the trees in R (roots are nodes in C2). We are
    # looking for the longest branches in the tree
    #-----
    for root_id in C2:
        R = snap.GetBfsTree(G2, root_id, True, False)

        # We will remove the elements of R from G2. What remains
        # in G2 is T
        for n in R.Nodes():
            G2.DelNode(n.GetId())

        B = FindLongestBranchInTree(R, root_id)
        if (len(B) > len(B1)):
            # If we found the best branch in the current tree, also
            # need to check if the second-best branch is in the
            # same tree
            B2 = B1
            B1 = B

        # Need to delete the previously identified branch, up to
        # the nearest branching point (since a branch point
        # represents another "chain" that we might save)
        for n_id in B1:
            n = R.GetNI(n_id)
            children = list(n.GetOutEdges())
            # Note that we are destroying the branch from leaf on
            # up. Hence, a single strand should always have
            # len(children) == 0
            if (len(children) > 0):
                break
            R.DelNode(n_id)

        B = FindLongestBranchInTree(R, root_id)
        B = [nid for nid in B if nid not in B1] # Don't double count
        if (len(B) > len(B2)):
            B2 = B
        elif (len(B) > len(B2)):
            B2 = B
    #print "B1: {}".format(B1)
    #print "B2: {}".format(B2)

    # If we did not find any interesting nodes to save, then just
    # pick an node at random (makes sense to take nodes not in
    # C2, A, or S. I'll proceed with blacklisting A only, per the

```

```

# problem instructions
if (len(B1)==0):
    #print "Used random selection for B1"
    blacklist = []
    #blacklist.extend(C2)
    blacklist.extend(A)
    #blacklist.extend(S)

    rnd_id = G.GetRndNid()
    while (rnd_id in blacklist):
        rnd_id = G.GetRndNid()

    B1 = [rnd_id]

# Next, we consider the trees in T (G2). First, we segment the
# graph into connected components. In each component, we
# need to look for the maximal chain that we can save by
# placing two anchors at the chain endpoints.
#-----
B3 = []

wccs = snap.TCnComV()
snap.GetWccs(G2, wccs)
for wcc in wccs:
    # We begin by forming a tree with a randomly chosen node
    # in the wcc
    root_id = wcc.GetRndNid()
    T = snap.GetBfsTree(G2, root_id, True, False)
    B = FindLongestBranchInTree(T, root_id)

    # We need to regenerate the tree from the leaf of the
    # deepest branch in order to guarantee we have the
    # longest possible chain
    if (len(B) > 0):
        root_id = B[0]
        T = snap.GetBfsTree(G2, root_id, True, False)
        B = FindLongestBranchInTree(T, root_id)
        B.append(root_id) # Note that B includes both endpoint nodes

        if (len(B) > len(B3)):
            B3 = B
#print "B3: {}".format(B3)

# Finally, we decide for the best option
if ((len(B1)+len(B2) > len(B3)) or (b==1)): # Choose 1 solution
    A.append(B1[0])
    S.extend(B1[1:])
    b = b-1

    # Bump up the second possible path
    B1 = B2
    B2 = []
else:
    A.append(B3[0])
    A.append(B3[-1])
    S.extend(B3[1:-1])
    b = b-2

#print "A: {}".format(A)
#print "S: {}".format(S)

# Compute the final results
equilibrium_set = []
equilibrium_set.extend(C2)
equilibrium_set.extend(A)
equilibrium_set.extend(S)

'''
print "* Final results for b={}".format(b_max)
print "C2: {}".format(C2)
print "A: {}".format(A)
print "S: {}".format(S)
print "Equilibrium set (size {}): {}".format(len(equilibrium_set), equilibrium_set)

```

```
'''  
print "{}, {}".format(b_max, len(equilibrium_set))
```