```
# Tony Hyun Kim
# 2013 10 22
# CS 224w, PS 2, Problem 3

import numpy as np
import snap

# Load the graphs
#-------------------------------------------------------------
G1 = snap.LoadEdgeList(snap.PUNGraph, "g1.edgelist", 0, 1)
G2 = snap.LoadEdgeList(snap.PUNGraph, "g2.edgelist", 0, 1)

# Initialize the voting state
#   I will use the following integer encoding:
#       0: Undecided
#       1: Candidate A
#      -1: Candidate B
#-------------------------------------------------------------
G = G1

# I decided to use a numpy vector for holding state, because
#   there seems to be issues with Snap.py attributes
N = 10000 # Number of nodes
state = np.zeros((N,1), int)

# The intial voting state is based on the last digit of nid
init_vote = {0: 1,
             1: 1,
             2: 1,
             3: 1,
             4: -1,
             5: -1,
             6: -1,
             7: -1,
             8: 0,
             9: 0}
for nid in range(N):
    state[nid] = init_vote[nid % 10]

tiebreak = 1 # Global variable for breaking ties

for day in range(1,11):

    state_next = np.zeros((N,1), int)

    # Iterate over all nodes in increasing Id
    for ni_id in range(N):
        # Only the initially undecided voters get modified
        if (init_vote[ni_id % 10] != 0):
            state_next[ni_id] = state[ni_id]
        else: # Undecided voter
            ni = G.GetNI(ni_id)
            friends_votes = 0
            for nj_id in ni.GetOutEdges():
                # If possible, use values from current iteration
                if (nj_id < ni_id):
                    friends_votes += state_next[nj_id]
                else:
                    friends_votes += state[nj_id]

            if (friends_votes == 0):
                state_next[ni_id] = tiebreak
                tiebreak *= -1
            elif (friends_votes > 0):
                state_next[ni_id] = 1
            else:
                state_next[ni_id] = -1

    # Update state
    state = state_next
    print "At end of day {0:02d}, sum(state)={1:d}".format(day, np.sum(state))
```

```
# Tony Hyun Kim
# 2013 10 22
# CS 224w, PS 2, Problem 3

import numpy as np
import snap

# Load the graphs
#-------------------------------------------------------------
G1 = snap.LoadEdgeList(snap.PUNGraph, "g1.edgelist", 0, 1)
G2 = snap.LoadEdgeList(snap.PUNGraph, "g2.edgelist", 0, 1)

# Initialize the voting state
#   I will use the following integer encoding:
#       0: Undecided
#       1: Candidate A
#      -1: Candidate B
#-------------------------------------------------------------
G = G2

# I decided to use a numpy vector for holding state, because
#   there seems to be issues with Snap.py attributes
N = 10000 # Number of nodes
state = np.zeros((N,1), int)

# The intial voting state is based on the last digit of nid
init_vote = {0: 1,
             1: 1,
             2: 1,
             3: 1,
             4: -1,
             5: -1,
             6: -1,
             7: -1,
             8: 0,
             9: 0}

# The following {id: vote} dict has the highest priority in
#   determining vote pattern
overrides = {}
for k in range(3000,3090):
    overrides[k] = 1

for nid in range(N):
    if nid in overrides:
        state[nid] = overrides[nid]
    else:
        state[nid] = init_vote[nid % 10]

tiebreak = 1 # Global variable for breaking ties

for day in range(1,11):

    state_next = np.zeros((N,1), int)

    # Iterate over all nodes in increasing Id
    for ni_id in range(N):
        if ni_id in overrides: # "Die-hard" voters
            state_next[ni_id] = overrides[ni_id]
        elif(init_vote[ni_id % 10] != 0):
            state_next[ni_id] = init_vote[ni_id % 10]
        else: # Undecided voter
            ni = G.GetNI(ni_id)
            friends_votes = 0
            for nj_id in ni.GetOutEdges():
                # If possible, use values from current iteration
                if (nj_id < ni_id):
                    friends_votes += state_next[nj_id]
                else:
                    friends_votes += state[nj_id]

            if (friends_votes == 0):
                state_next[ni_id] = tiebreak
```

```
                tiebreak *= -1
            elif (friends_votes > 0):
                state_next[ni_id] = 1
            else:
                state_next[ni_id] = -1

    # Update state
    state = state_next
    print "At end of day {0:02d}, sum(state)={1:d}".format(day, np.sum(state))
```

```
# Tony Hyun Kim
# 2013 10 22
# CS 224w, PS 2, Problem 3

import numpy as np
import snap

# Load the graphs
#-------------------------------------------------------------
G1 = snap.LoadEdgeList(snap.PUNGraph, "g1.edgelist", 0, 1)
G2 = snap.LoadEdgeList(snap.PUNGraph, "g2.edgelist", 0, 1)

# Initialize the voting state
#   I will use the following integer encoding:
#       0: Undecided
#       1: Candidate A
#      -1: Candidate B
#-------------------------------------------------------------
G = G2

# I decided to use a numpy vector for holding state, because
#   there seems to be issues with Snap.py attributes
N = 10000 # Number of nodes
state = np.zeros((N,1), int)

# The intial voting state is based on the last digit of nid
init_vote = {0: 1,
             1: 1,
             2: 1,
             3: 1,
             4: -1,
             5: -1,
             6: -1,
             7: -1,
             8: 0,
             9: 0}

# The following {id: vote} dict has the highest priority in
#   determining vote pattern
'''
# Hit list for G1
overrides = {354: 1,
             896: 1,
             7035: 1,
             804: 1,
             1878: 1,
             3685: 1,
             2704: 1,
             3307: 1,
             7137: 1,
            }
'''
# Hit list for G2
overrides = {17: 1,
             16: 1,
             15: 1,
             6: 1,
             26: 1,
             18: 1,
             14: 1,
             5: 1,
             4: 1,
            }

for nid in range(N):
    if nid in overrides:
        state[nid] = overrides[nid]
    else:
        state[nid] = init_vote[nid % 10]

tiebreak = 1 # Global variable for breaking ties

for day in range(1,11):
```

```
    state_next = np.zeros((N,1), int)

    # Iterate over all nodes in increasing Id
    for ni_id in range(N):
        if ni_id in overrides: # "Die-hard" voters
            state_next[ni_id] = overrides[ni_id]
        elif(init_vote[ni_id % 10] != 0):
            state_next[ni_id] = init_vote[ni_id % 10]
        else: # Undecided voter
            ni = G.GetNI(ni_id)
            friends_votes = 0
            for nj_id in ni.GetOutEdges():
                # If possible, use values from current iteration
                if (nj_id < ni_id):
                    friends_votes += state_next[nj_id]
                else:
                    friends_votes += state[nj_id]

            if (friends_votes == 0):
                state_next[ni_id] = tiebreak
                tiebreak *= -1
            elif (friends_votes > 0):
                state_next[ni_id] = 1
            else:
                state_next[ni_id] = -1

    # Update state
    state = state_next
    print "At end of day {0:02d}, sum(state)={1:d}".format(day, np.sum(state))
```

```
# Tony Hyun Kim
# 2013 10 22
# CS 224w, PS 2, Problem 3

import numpy as np
import snap

# Load the graphs
#-------------------------------------------------------------
G1 = snap.LoadEdgeList(snap.PUNGraph, "g1.edgelist", 0, 1)
G2 = snap.LoadEdgeList(snap.PUNGraph, "g2.edgelist", 0, 1)

# Determine the highest rollers of the graph
#-------------------------------------------------------------
init_vote = {0: 1,
             1: 1,
             2: 1,
             3: 1,
             4: -1,
             5: -1,
             6: -1,
             7: -1,
             8: 0,
             9: 0}

G = G2

high_rollers = []
while (len(high_rollers) < 20):
    nid = snap.GetMxDegNId(G)

    # Target nodes that were not hard-wired for candidate A
    #if (True):
    if (init_vote[nid % 10] != 1):
        high_rollers.append(nid)
        n = G.GetNI(nid)
        print "node {} has degree {}".format(nid, n.GetOutDeg())

    G.DelNode(nid)
```