Social Network Programming Assignment 2
By Neelu Verma
(MP19AI002)

February 2, 2022

1 Coding Assignment 2: Find k-core

Please select a network of your choice as previous assignment and do the following:

- Calculate Closeness Centrality
- Calculate Betweenness Centrality
- For a random k between 0-20 fill all the k-cores in the network
- Report the number and size of the maximum and minimum clique

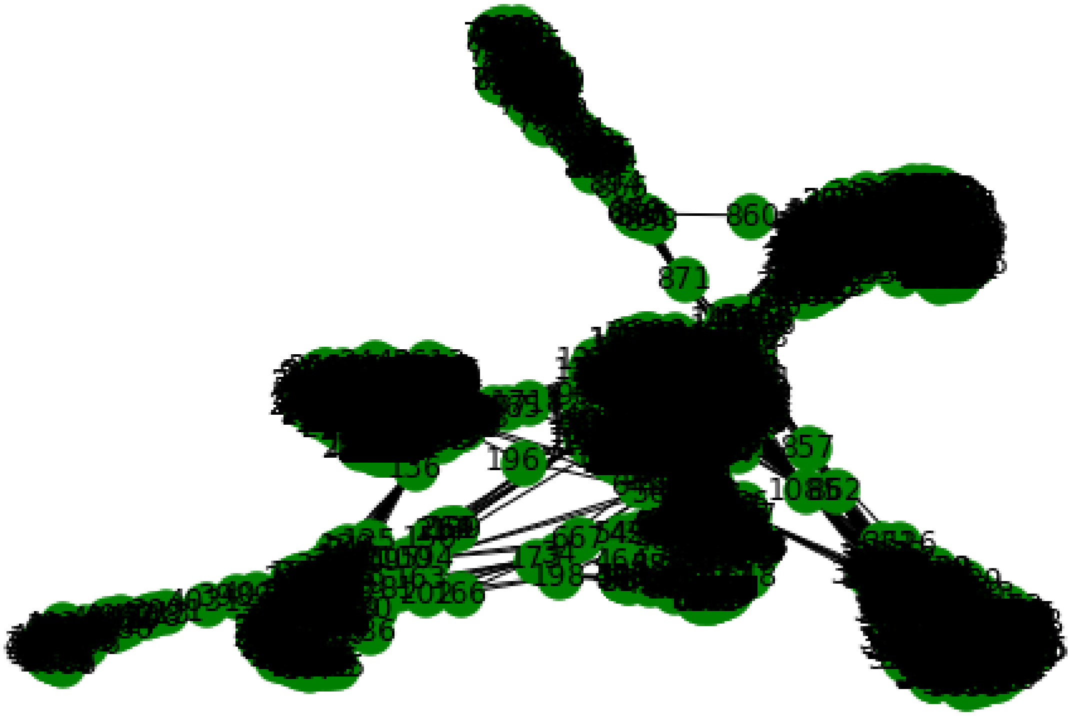
For 1, 2 and 3 do not use the exact functions available in NetworkX. However, you can use function available for finding distance/shortest distance.

[1]:
```
# basic information about graph
import networkx as nx
import matplotlib.pyplot as plt

G =nx.read_edgelist("facebook_combined.txt",create_using = nx.Graph(),␣
 ↪nodetype=int)
print(nx.info(G))
```

```
Name:
Type: Graph
Number of nodes: 4039
Number of edges: 88234
Average degree:  43.6910
```

[2]:
```
#Drawing a graph with all nodes
nx.draw(G, with_labels = True, node_color ='green' )
```



2 1. Definitions for Calculate Closeness Centrality:

Closeness centrality is a measure of the average shortest distance from each vertex to each other vertex. Specifically, it is the inverse of the average shortest distance between the vertex and all other vertices in the network. The formula is 1/(average distance to all other vertices).

[12]:
```
# Method 1. Calculate Closeness Centrality using diferent method to compare␣
 ↪results:
N=list(G.nodes)
E=list(G.edges)

path=dict(nx.algorithms.shortest_paths.unweighted.all_pairs_shortest_path(G))
close_cen=[]
for i in range(len(nodes)):
    suml=0
    for j in range(len(nodes)):
        if(i!=j):
            suml=suml+len(path[i][j])-1
    close_cen.append(1/suml)

closeness_centrality=close_cen
print(len(closeness_centality))
```

```
print(closeness_centrality)
```

note: arrays contain lots of values of all nodes, I have removed some to reduced the size of pdf. for more detail check ipynb file

```
[8.750437521876094e-05, 6.472910868017347e-05, 6.469979296066253e-05,
6.472910868017347e-05, 6.469979296066253e-05, 6.471235358830001e-05,
6.46830530401035e-05, 6.804572672836147e-05, 6.469142191745374e-05,
6.48971380362126e-05, 6.469979296066253e-05, 6.466214031684449e-05,
6.466214031684449e-05, 6.47878198898607e-05, 6.472073004983496e-05,
6.466214031684449e-05, 6.469560716827328e-05, 6.471235358830001e-05,
7.916402786573781e-05, 8.367500627562547e-05, 7.92016473942658e-05,
 7.050197405527355e-05,
6.998880179171333e-05, 6.998390370214851e-05, 6.996921354603975e-05,
7.004763239002522e-05, 7.005744710662743e-05, 7.004763239002522e-05,
7.026419336706015e-05, 7.047216349541931e-05, 7.018528916339136e-05,
6.997900629811057e-05, 7.786342754808066e-05, 5.932253663166637e-05,
5.9269796111901376e-05, 6.88089176357256e-05, 5.925925925925926e-05,
5.9396531242575434e-05, 5.9315499139925264e-05, 5.925574780753733e-05,
4.5583006655118975e-05, 4.5589240939138365e-05,
4.5578851412944395e-05, 4.556431402925229e-05, 4.5574696928265424e-05,
4.557054320087495e-05, 4.558092893933178e-05, 4.5566390230565937e-05,
4.558092893933178e-05, 4.5562238017131404e-05, 5.566069241901369e-05,
4.556431402925229e-05, 4.557261996992207e-05, 4.558508456033186e-05,
4.5562238017131404e-05, 4.55684666210982e-05, 4.5578851412944395e-05,
4.5574696928265424e-05, 4.55767740759309e-05, 4.55767740759309e-05,
4.5583006655118975e-05, 4.5562238017131404e-05, 4.5597555970999955e-05,
4.5562238017131404e-05, 4.55684666210982e-05, 4.5578851412944395e-05,
4.5574696928265424e-05, 4.556431402925229e-05, 4.556431402925229e-05,
4.5599635202918376e-05, 5.5682387660782896e-05, 4.556431402925229e-05,
4.5566390230565937e-05, 4.556431402925229e-05, 4.5562238017131404e-05,
4.556431402925229e-05, 4.55684666210982e-05, 4.5578851412944395e-05]
```

[2]:
```
#Method 2. Calculate Closeness Centrality using diferent method to compare␣
↪results:
def closeness_centrality(G, u=None, distance=None, normalized=True):
    r
    if distance is not None:

        # use Dijkstra's algorithm with specified attribute as edge weight
        path_length = functools.partial(nx.single_source_dijkstra_path_length,
                                        weight=distance)
    else:
        path_length = nx.single_source_shortest_path_length

    if u is None:
```

```
        nodes = G.nodes()
    else:
        nodes = [u]
    closeness_centrality = {}
    for n in nodes:
        sp = path_length(G,n)
        totsp = sum(sp.values())
        if totsp > 0.0 and len(G) > 1:
            closeness_centrality[n] = (len(sp)-1.0) / totsp

            # normalize to number of nodes-1 in connected part
            if normalized:
                s = (len(sp)-1.0) / ( len(G) - 1 )
                closeness_centrality[n] *= s
        else:
            closeness_centrality[n] = 0.0
    if u is not None:
        return closeness_centrality[u]
    else:
        return closeness_centrality
c=nx.closeness_centrality(G)
print(c)
```

```
{'0': 0.35334266713335666, '1': 0.2613761408505405, '2': 0.26125776397515527,
'3': 0.2613761408505405, '4': 0.26125776397515527, '5': 0.2613084837895554, '6':
0.26119016817593793, '7': 0.2747686445291236, '8': 0.2612239617026782, '9':
0.2620546433902265, '10': 0.26125776397515527, '11': 0.26110572259941806, '12':
0.26110572259941806, '13': 0.26161321671525756, '14':
'1805': 0.31546875, '1806': 0.3357166611240439, '1807': 0.3161851068827813,
'1808': 0.33416087388282023, '1809': 0.3177275946179873, '1810':
0.23928888888888888, '3971': 0.2398716882499703, '3972': 0.2394591709660203,
'3973': 0.23931725241510104, '3974': 0.239218009478673, '3975':
0.23941657773034508, '3976': 0.23941657773034508, '3977': 0.23928888888888888,
'3978': 0.2392605320850862, '3979': 0.239373999644318, '3981':
0.184039013718609, '3982': 0.1840893549122407, '3983': 0.18398870005012075,
'4021': 0.1840641808733704, '4022': 0.1839803171131766, '4023':
0.18412293101089783, '4024': 0.1839803171131766, '4025': 0.18400546821599453,
'4026': 0.18404740200546946, '4027': 0.18403062619633578, '4028':
0.18398870005012075, '4029': 0.18398870005012075, '4030': 0.18413132694938442,
'4032': 0.18398870005012075, '4033': 0.18399708375102525, '4034':
0.18398870005012075, '4035': 0.1839803171131766, '4036': 0.18398870005012075,
'4037': 0.18400546821599453, '4038': 0.18404740200546946}
```

3 2. Calculate Betweenness Centrality:

To calculate betweenness centrality, you take every pair of the network and count how many times a node can interrupt the shortest paths (geodesic distance) between the two nodes of the pair

[13]:
```
#Method 1: Calculate Betweenness Centrality from scratch
from itertools import combinations
Gbc=nx.gnp_random_graph(500, 0.2, seed=None, directed=False)
L = list(Gbc.nodes)
pairs=[" ".join(map(str, comb)) for comb in combinations(L, 2)]
path=dict(nx.algorithms.shortest_paths.unweighted.all_pairs_shortest_path(Gbc))


bc=[]
div=len(pairs)
for i in range(len(L)):

    cnt=0
    for j in pairs:
        st = [int(x) for x in j.split()]
        if(i not in st):
            path_node=path[st[0]][st[1]]
            if (i in path_node):
                cnt=cnt+1


    bc.append(cnt/div)
print(bc)
```

```
[0.026140280561122244, 0.031342685370741483, 0.029138276553106213,
0.026236472945891782, 0.030020040080160320, 0.026340681362725452,
0.022188376753507013, 0.022076152304609218, 0.021402805611222445,
0.0, 0.0, 8.016032064128256e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 8.016032064128256e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0]
```

[3]:
```
# Method 2:  Calculate Betweenness Centrality using BFS and Dijiksta's both
def betweenness_centrality(G, k=None, normalized=True, weight=None,
                           endpoints=False, seed=None):
    r
    betweenness = dict.fromkeys(G, 0.0)
    if k is None:
        nodes = G
    else:
        random.seed(seed)
        nodes = random.sample(G.nodes(), k)
    for s in nodes:
        if weight is None:  # use BFS
```

```
            S, P, sigma = _single_source_shortest_path_basic(G, s)
        else:  # use Dijkstra's algorithm
            S, P, sigma = _single_source_dijkstra_path_basic(G, s, weight)
        if endpoints:
            betweenness = _accumulate_endpoints(betweenness, S, P, sigma, s)
        else:
            betweenness = _accumulate_basic(betweenness, S, P, sigma, s)
    betweenness = _rescale(betweenness, len(G), normalized=normalized,
                           directed=G.is_directed(), k=k)
    return betweenness

d=nx.betweenness_centrality(G)
print(d)
```

```
{'0': 0.14630592147442917, '1': 2.7832744209034606e-06, '2':
7.595021178512074e-08, '3': 1.6850656559280464e-06, '4': 1.8403320547933104e-07,
'5': 2.205964164092193e-06, '6': 2.4537760730577472e-08, '7':
0.0001702984836730339, '8': 2.7604980821899654e-07, '9': 1.6454236303026905e-05,
'10': 4.986739552037655e-08, '11': 0.0, '12': 0.0, '13': 1.7622717578436846e-06,
'14': 5.582871686568508e-07, '15': 0.0, '16': 1.9979459275532697e-07, '17':
4.1066669000480344e-07, '18': 0.0, '19': 5.062957964075819e-06, '20':
6.793693332142838e-07, '21': 0.0009380243844653233, '22': 6.703002200833232e-07,
'23': 6.860348937590618e-06, '24': 1.3673472422981514e-07, '25':
5.38808313945586e-05, '26': 1.935436798204632e-05, '27': 3.067220091322184e-08,
'28': 3.812160659244892e-07, '29': 1.3954817951917517e-06, '30':
1.3694627409316544e-06, '31': 4.932641252790837e-06, '32': 0.0, '33': 0.0, '34':
9.8487861751963e-07, '3963': 9.610622952809509e-08, '3964': 2.1661389889382e-06,
8.721129126326074e-07, '4005': 0.0,
'4006': 0.0, '4007': 0.0, '4008': 0.0, '4009': 2.4683818830164235e-07, '4010':
0.0, '4012': 0.0, '4013': 1.0224066971073946e-07, '4014':
1.4651087969548963e-06, '4015': 0.0, '4016': 0.0, '4017': 6.923153920412929e-07,
'4018': 2.9795852315701213e-08, '4019': 2.0769461761238785e-07, '4020':
5.602788700148523e-07, '4021': 6.350606170032787e-07, '4022': 0.0, '4023':
4.477411042832454e-06, '4024': 0.0, '4025': 0.0, '4026': 3.968398565772558e-07,
'4027': 5.766373771685704e-07, '4028': 0.0, '4029': 0.0, '4030':
4.542114780949394e-06, '4032': 0.0, '4033': 0.0, '4034': 0.0, '4035': 0.0,
'4036': 0.0, '4037': 7.156846879751761e-08, '4038': 6.338921522065847e-07}
```

4 3.For a random k between 0-20 fill all the k-cores in the network:

[44]:
```
#. 3 k-core
k=4

for i in range(0,21):
    size=100
    fb= G
```

```
    while size>0:

        remove = [node for node,degree in dict(fb.degree()).items() if degree <␣
 ↪i]
        size=len(remove)
        fb.remove_nodes_from(remove)


    nn=nx.number_connected_components(fb)
    print(nn)


cnodes=fb.number_of_nodes()
cedges=fb.number_of_edges()

kcore_size=nx.connected_components(fb)
print("Size of each connected component:",list(kcore_size))
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
Size of each connected component: [{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141,
```

```
142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157,
158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173,
174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189,
190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205,
206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221,
222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237,
238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253,
254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269,
270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,
286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301,
302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317,
318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333,
334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349,
350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365,
366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381,
382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397,
398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413,
414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429,
430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445,
446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461,
462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477,
478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493,
494, 495, 496, 497, 498, 499}]
```

5 4. Report the number and size of the maximum and minimum clique:

[40]:

```
#4
G=nx.gnp_random_graph(500, 0.2, seed=None, directed=False)
edge=G.edges()

nodes=G.number_of_nodes()

edges=G.number_of_edges()

cliques=nx.find_cliques(G)

cliques = list(cliques)

print("Total number of cliques: ",len(cliques))

max_index=4000
min_index=4000

maxc=0
minc=4000
```

```
for i in range(len(cliques)):
    l=len(cliques[i])
    if l>maxc:
        maxc=l
        max_index=i

    if l<minc:
        minc=l
        min_index=i

max_cliques=[]
min_cliques=[]

for i in range(len(cliques)):

    l=len(cliques[i])
    if l==maxc:

        max_cliques.append(cliques[i])
    if l==minc:
        min_cliques.append(cliques[i])

print("Size of max clique:",len(max_cliques))
print("Size of min clique:",len(min_cliques))

print("----using inbuit function-----")
#number of max clique
print("number of maximum clique:", nx.graph_number_of_cliques(G))

#size of max clique
print("Size of max clique:", max(nx.node_clique_number(G).values()))
```

```
Total number of cliques:  97932
Size of max clique: 6
Size of min clique: 3273
----using inbuit function-----
number of maximum clique: 97932
Size of max clique: 7
```

[11]:

```
#######################################--------Finish----######################################
```

6 Thankyou