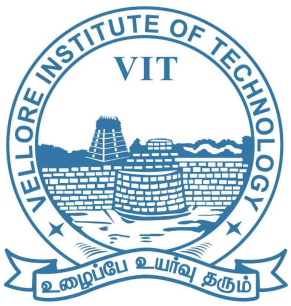


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Name*: *Anushree Paul

Register No*: *22MCB0019

Subject*: Social Network Analytics *Lab

***Subject Code*: MCSE618L**

Faculty Name: Prof. DURGESH KUMAR

E-mail: anushree.paul2022@vitstudent.ac.in

Submission Date: 01-june-2023

Lab report :

Topic: Community detection

```
#implement community detection algorithm on a social network
import networkx as nx
def edge_to_remove(g):
  d1 = nx.edge_betweenness_centrality(g)
  list_of_tuples = list(d1.items())

  sorted(list_of_tuples, key = lambda x:x[1], reverse = True)

  # Will return in the form (a,b)
  return list_of_tuples[0][0]

def girvan(g):
  a = nx.connected_components(g)
  lena = len(list(a))
  print (' The number of connected components are ', lena)
  while (lena == 1):

    # We need (a,b) instead of ((a,b))
    u, v = edge_to_remove(g)
    g.remove_edge(u, v)

    a = nx.connected_components(g)
    lena=len(list(a))
    print (' The number of connected components are ', lena)

  return a

# Driver Code
g = nx.barbell_graph(5,0)
a = girvan(g)
print ('Barbell Graph')

for i in a:
  print (i.nodes())
  print ('.............')
```

```
g1 = nx.karate_club_graph()
a1 = girvan(g1)

print ('Karate Club Graph')
for i in a1:
  print (i.nodes())
  print ('.............')
```

```
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  2
Barbell Graph
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  1
 The number of connected components are  2
Karate Club Graph
```

```
import networkx as nx
G = nx.barbell_graph(5,1)
communities_generator = nx.community.girvan_newman(G)
top_level_communities = next(communities_generator)
next_level_communities = next(communities_generator)
sorted(map(sorted,next_level_communities))
[[4,5,6,7,8,9],[10],[12,13,18,19,20]]
```

```
[[4, 5, 6, 7, 8, 9], [10], [12, 13, 18, 19, 20]]
```

```
G = nx.complete_graph(5)
k5 = nx.convert_node_labels_to_integers(G, first_label=2)
G.add_edges_from(k5.edges())
c = list(nx.community.k_clique_communities(G,4))
sorted(list(c[0]))
```

```
[0, 1, 2, 3, 4, 5, 6]
```

```
list(nx.community.k_clique_communities(G,6))
```

```
[]
```

```
#greedy modularity communities
G = nx.karate_club_graph()
c = nx.community.greedy_modularity_communities(G)
sorted(c[0])
```

```
[8, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]
```

```
#naive_greedy_modularity_communities
G = nx.karate_club_graph()
c = nx.community.naive_greedy_modularity_communities(G)
sorted(c[2])
```

```
[0, 4, 5, 6, 10, 11, 16, 19]
```

```
#louvain communities
import networkx as nx
G = nx.petersen_graph()
nx.community.louvain_communities(G,seed = 789)
```

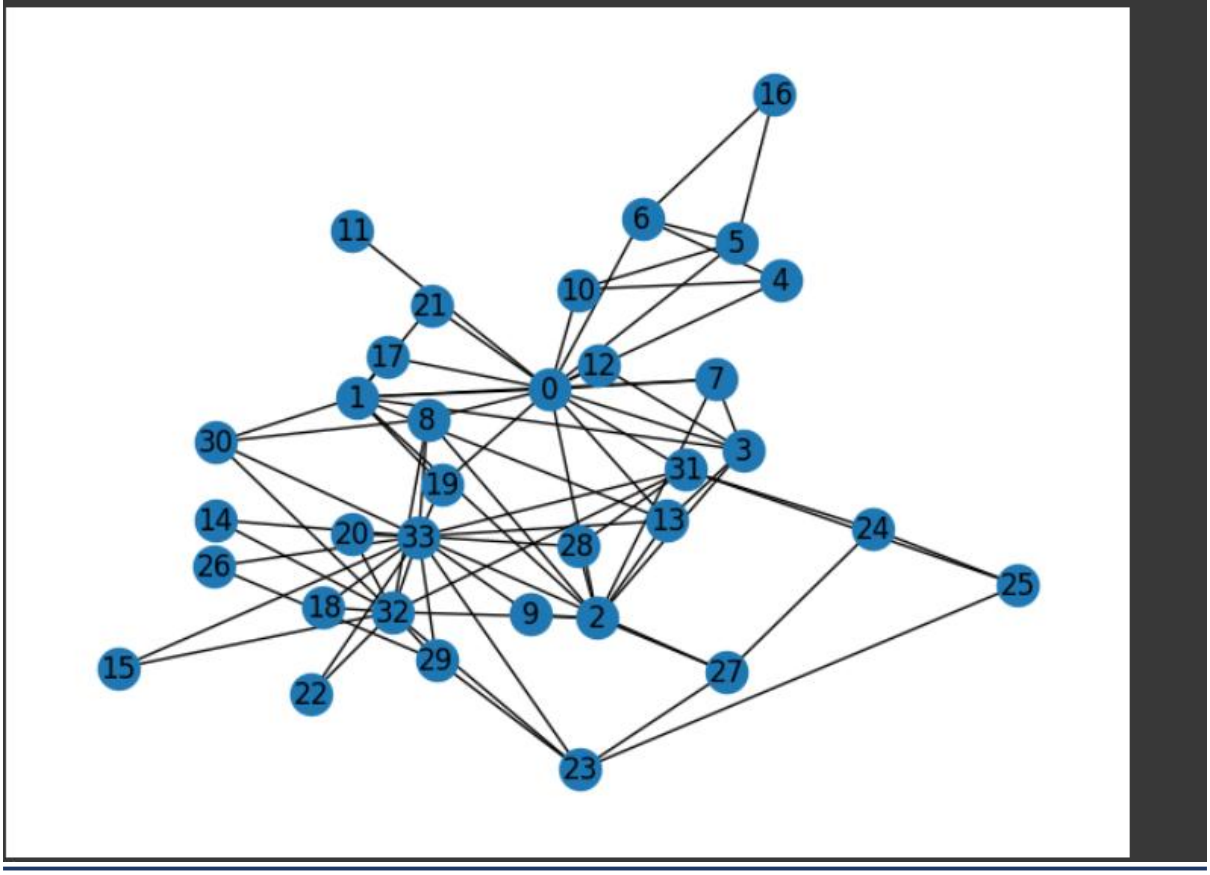
```
[{0, 5}, {1, 6}, {3, 8}, {2, 7}, {4, 9}]
```

```
#modularity
G = nx.barbell_graph(3,0)
nx.community.modularity(G,[{0,1,2},{3,4,5}])
```

```
0.35714285714285715
```

```
import matplotlib.pyplot as plt
%matplotlib inline

import networkx as nx
G = nx.karate_club_graph()
nx.draw_kamada_kawai(G, with_labels = True)
```



```
#Calculating the betweenness centrality
btw_centrality = nx.algorithms.centrality.edge_betweenness_centrality(G)
# Sorting based on the betweenness centraliy and displaying the first 10 edges.
sorted(btw_centrality.items(), key = lambda item:item[1], reverse = True)[0:10]
```

```
[((0, 31), 0.1272599949070537),
 ((0, 6), 0.07813428401663695),
 ((0, 5), 0.07813428401663694),
 ((0, 2), 0.0777876807288572),
 ((0, 8), 0.07423959482783014),
 ((2, 32), 0.06898678663384543),
 ((13, 33), 0.06782389723566191),
 ((19, 33), 0.05938233879410351),
 ((0, 11), 0.058823529411764705),
 ((26, 33), 0.0542908072319837)]
```

```
def girwan_newman(G, no_of_components_to_split):
 while(no_of_components_to_split > nx.algorithms.components.number_connected_component):
 # Calculate the betweenness centrality
  btw_centrality = nx.algorithms.centrality.edge_betweenness_centrality(G)
 # sort based on betweenness centrality
 sorted_edges = sorted(btw_centrality.items(), key = lambda item:item[1], reverse=True)
 print('Removing the edge', sorted_edges)
 # remove edge which has highest centrality
 G.remove_edge(*sorted_edges[0])
 # Check if graph is split
 if(no_of_components_to_split <= nx.algorithms.components.number_connected_component):
 # Plot the graph with both the nodes having different colors
  nx.draw_spring(G, with_labels=True)
 # return list of nodes in each community
 list_of_nodes = [c for c in sorted(nx.connected_components(G), key=len, reverse=True)]
 return list_of_nodes
```

```
import networkx as nx
def girvan_newman(G, most_valuable_edge=None):
  G = nx.path_graph(10)
  comp = girvan_newman(G)
  tuple(sorted(c) for c in next(comp))
  ([0, 1, 2, 3, 4], [5, 6, 7, 8, 9])
```

```
from networkx import edge_betweenness_centrality as betweenness
def most_central_edge(G):
  centrality = betweenness(G, weight="weight")
  return max(centrality, key=centrality.get)
  G = nx.path_graph(10)
  comp = girvan_newman(G, most_valuable_edge=most_central_edge)
  tuple(sorted(c) for c in next(comp))
  ([0, 1, 2, 3, 4], [5, 6, 7, 8, 9])
```

```
from networkx import edge_betweenness_centrality
from random import random
def most_central_edge(G):
 centrality = edge_betweenness_centrality(G)
 max_cent = max(centrality.values())
# Scale the centrality values so they are between 0 and 1,
# and add some random noise.
 centrality = {e: c / max_cent for e, c in centrality.items()}
# Add some random noise.
 centrality = {e: c + random() for e, c in centrality.items()}
 return max(centrality, key=centrality.get)
 G = nx.path_graph(10)
 comp = girvan_newman(G, most_valuable_edge=most_central_edge)
```