

After loading the adjacency list *vitevitch.adjlist*. We will be using the *networkx* and *matplotlib.pyplot* python libraries to help us analyze the network. We will also be using functions from Assignment 1 and 2 and other code written by ourselves to further analyzed the properties of the network. Appendix 1 shows all of the code that was written for Assignment 3.

Table 1 shows a mapping of all nodes and their corresponding degree values.

Node	Degree	Node	Degree
bog	5	dog	7
fog	4	log	4
hog	4	bag	2
dig	2	dug	2
dawn	1	bat	2
cattle	1	cat	18
cot	5	kite	5
kit	5	coat	5
cut	5	cad	4
calf	4	can	4
cab	4	catch	1
chat	6	that	6
hat	6	rat	6
fat	6	gnat	6

Table 1: Nodes and their Degree

From a glance at the table, the degrees range from 1 to 7 excluding 3, which is not present. Cat is an exception however, as it has a degree of 18, and is much larger then the others. Since we have the degrees of all nodes we are able to represent them in a histogram format to see the frequency easier, which is represented by Figure 1

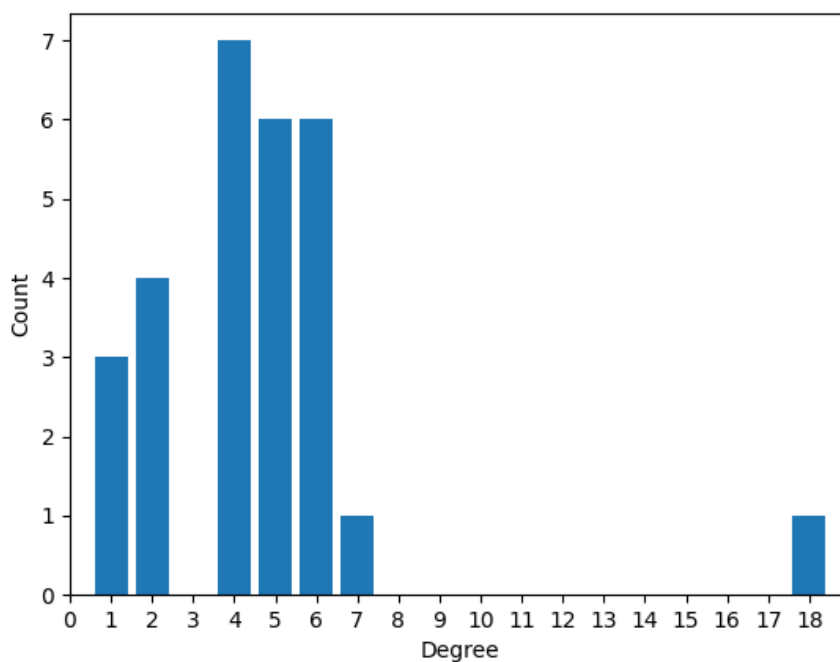


Figure 1: Degree Frequency

From Figure 1 we can tell that the maximum degree is indeed 18 and belongs to our node 'cat'.

We will now examine the "cat" node further and look at compare the betweenness, and closeness centrality with the means of each. Table 2 shows the details of these comparisons:

	Cat Node	Averages of Network
Degree	18	4.642857
Betweenness	0.783476	0.081705
Closeness centrality	0.5	0.333327

Table 2: Comparison of Cat Node with Network

The degree of "cat" is four times approximately 3.8 times higher then the average of the network. The betweenness is approximately 9.6 times higher then the average, and closeness is approximately 1.5 times higher in "cat".

Due to "cat" having a high betweenness centrality and a significantly high degree compared to other nodes it can be considered to be a hub. Cat also has the highest closeness centrality as it is the closest one to all other nodes, this is mainly due to the different in degrees for each node shown in Figure 1 and Table 1.

From the Samuel Arbesman reading, our assumptions from Assignment 1 were confirmed, where each node is connected if they differ by one phoneme. Additionally it verified that there was usually one large group of nodes (a hub) and other smaller islands. In the results section of the paper, it mentioned typically the giant component contained 80 to 90 percent of vertices, however in present networks the proportion of vertices in the giant component was actually smaller with some networks having less than 50% in our giant component. In our example 18/28 vertices were connected to our cat hub which is roughly 60 percent. To look for robustness, they removed vertices at random and in decreasing degree order. If vertices were removed in a random order, the mean shortest path remained constant whereas if removed in order of degree the shortest path length increased dramatically. I preformed a similar experiment, where I removed 5 percent of the vertices (2 vertices) randomly. to see how it impacted the average path. As we see in Table 3, our average path length was also relatively a constant at 3.17. However, one thing to note is that when removing we have to ensure the graph is still connected.

Removing Randomly	
Trial 1	3.1723076923076925
Trial 2	3.163076923076923
Trial 3	3.1753846153846155

Table 3: Removing Nodes and Measuring Average Path

We could not remove degrees in decreasing order as removing cat results in our graph to be disconnected. However, I assume we would have a similar result, as cat is a hub and has a path of 1 to most vertices in the network. Removing vertexes who have the most connections will only increase our path lengths.

Our top 8 nodes with the most degrees were: 'cat', 'dog', 'chat', 'that', 'hat', 'rat', 'gnat', 'fat'. From the Assosrtative Mining portion of the reading it stated that "high degree vertices tended to be connected to each other." Table 4 shows whether or not our top 9 nodes are connected to each other or not where a "Y" means they are connected by an edge and "N" means they are not connected by an edge, and a blank means the path would be 0 as it doesn't connect to itself.

	cat	chat	that	hat	rat	gnat	fat	dog
cat		Y	Y	Y	Y	Y	Y	N
chat	Y		Y	Y	Y	Y	Y	N
that	Y	Y		Y	Y	Y	Y	N
hat	Y	Y			Y	Y		N
rat	Y	Y	Y	Y		Y	Y	N
gnat	Y	Y	Y	Y	Y		Y	N
fat	Y	Y	Y	Y	Y	Y		N
dog	N	N	N	N	N	N	N	

Table 4: Looking at if the top 8 vertex degrees are connected by edges

From this we can gather that our network also follows the same properties as the one in the reading, with dog being an exception.

Appendix 1 - Code Used

```
1 import matplotlib.pyplot as plt
2 import networkx as nx
3 import statistics
4 import random
5 from Assignments.A1.A1 import number_of_nodes, number_of_edges, highest_degree, lowest_degree,
6     average_degree, \
7     histogram_degrees
8 from Assignments.A2.A2 import shortest_path
9
10 def main():
11     graph = nx.read_adjlist('../Datasets/vitevitch.adjlist')
12     nx.draw(graph,
13             with_labels=True,
14             node_color='black',
15             node_size=18,
16             font_size=8,
17             verticalalignment='baseline',
18             edge_color='grey')
19     plt.show()
20     nodes = number_of_nodes(graph)
21     print(nodes)
22     edges = number_of_edges(graph)
23     max_degree = highest_degree(graph)
24     mean_degree = average_degree(nodes, edges)
25     histogram_degrees(graph)
26     node_degree_mapping = create_dictionary(graph)
27     node_max_between = between(graph)
28     max_closeness = closeness(graph, max_degree[0])
29     print(f"Our node that has a max degree is {max_degree} \n"
30           f"Our max degree is {graph.degree(max_degree[0])} \n"
31           f"Our graph has a mean degree of {mean_degree} \n"
32           f"Our max betweenness node is {node_max_between[0]} \n"
33           f"Our max value for between is {node_max_between[1]} \n"
34           f"Max closeness is {max_closeness}")
35
36     betweenness = nx centrality.betweenness centrality(graph)
37     betweenness_sequence = list(betweenness.values())
38     print('Mean betweenness:', statistics.mean(betweenness_sequence))
39     mean_closeness = closeness_mean(graph)
40     print(f"Mean closeness is {mean_closeness}")
41
42     # Create two nodes list
43     list_of_nodes = node_list_create(graph)
44     list_of_nodes_random = node_list_create(graph)
45     list_of_nodes = node_list_sort(list_of_nodes, graph, nodes)
46     list_of_nodes = list_of_nodes[0:8]
47     random.shuffle(list_of_nodes_random)
48     average_path_random = remove_nodes(list_of_nodes_random)
49
50     print(f"Removing two orders in random degree makes the path {average_path_random}")
51     print(f"Nodes in decreasing order of degree are {list_of_nodes}")
52     connected_print(graph, list_of_nodes)
53
54 def connected_print(G, nodes_list):
55     for i in range(0, len(nodes_list)):
56         for j in range(0, len(nodes_list)):
57             if i == j:
58                 pass
59             elif G.has_edge(nodes_list[i], nodes_list[j]):
60                 print(f"There exists an edge between {nodes_list[i]} {nodes_list[j]}")
61             print("\n \n")
62
63 def remove_nodes(nodes_list):
64     graph_r = nx.read_adjlist('../Datasets/vitevitch.adjlist')
65     for i in range(0, 2):
66         graph_r.remove_node(nodes_list[i])
67     return shortest_path(graph_r)
68
69 def node_list_sort(list_of_nodes, graph, nodes):
70     for i in range(0, nodes):
71         for j in range(i + 1, nodes):
72             if graph.degree(list_of_nodes[j]) > graph.degree(list_of_nodes[i]):
73                 tmp = list_of_nodes[j]
74                 list_of_nodes[j] = list_of_nodes[i]
75                 list_of_nodes[i] = tmp
76     return list_of_nodes
77
78 def node_list_create(G):
79     node_list = []
80     for node in G.nodes():
81         node_list.append(node)
82     return node_list
83
84 def closeness_mean(G):
85     total = 0
86     for node in G.nodes():
87         total = total + nx.closeness centrality(G, node)
88     total = total / number_of_nodes(G)
89     return total
```

```

90
91 def closeness(G, node):
92     return nx.closeness centrality(G, node)
93
94 def between(G):
95     return_list = []
96     betweenness = nx.centrality.betweenness centrality(G)
97     highest_betweenness_node = max(G.nodes, key=betweenness.get)
98     return_list.append(highest_betweenness_node)
99     return_list.append(betweenness[highest_betweenness_node])
100     return return_list
101
102 def create_dictionary(G):
103     return_dict = {}
104     for nodes in G.nodes():
105         return_dict[nodes] = G.degree(nodes)
106     return return_dict
107
108 if __name__ == '__main__':
109     main()

```