Homework #1

This homework is complete and will not be changed. The homework does not require a lot of writing, but may require a lot of thinking. It does not require a lot of processing power, but may require efficient programming. It accounts for 12.5% of the course grade. All questions and comments regarding the homework should be directed to Piazza.

Submission details

This homework is due on **March 23rd** at 2:00pm, while late days expire on **March 27th** at 1:00pm. The homework must be submitted as a hard-copy in the submission box in front of R 2.49 and also as an electronic version to eUcilnica. It can be prepared in either English or Slovene and either written by hand or typed on a computer. The hard-copy should include (1) this cover sheet with filled out time of the submission and signed honor code, (2) short answers to the questions, which can also demand proofs, tables, plots, diagrams and other, and (3) a printout of all the code required to complete the exercises. The electronic submission should include only (1) answers to the questions in a single file and (2) all the code in a format of the specific programming language. Note that hard-copies will be graded, while electronic submissions will be used for plagiarism detection. The homework is considered submitted only when both versions have been submitted. Failing to include this honor code in the submission will result in **10% deduction**. Failing to submit all the developed code to eUcilnica will result in **50% deduction**.

Honor code

The students are strongly encouraged to discuss the homework with other classmates and form study groups. Yet, each student must then solve the homework by herself or himself without the help of others and should be able to redo the homework at a later time. In other words, the students are encouraged to collaborate, but should not copy from one another. Referring to any solutions obtained from classmates, course books, previous years, found online or other, is considered an honor code violation. Also, stating any part of the solutions in class or on Piazza is considered an honor code violation. Finally, failing to name the correct study group members, or filling out the wrong date or time of the submission, is also considered an honor code violation. Honor code violation will not be tolerated. Any student violating the honor code will be reported to **faculty disciplinary committee** and vice dean for education.

Name & SID: Tara Patricija Bosil, 63160008

Study group: Miha Arh, Kristjan Šuler, Katja Logar, Luka Tavčer

Date & time: 23.03.2020, 13:55

I acknowledge and accept the honor code.

Signature: _____

1 Networkology (5 points)

1.1 Node degrees

Given networks are different. The differences are in both degree sequence k_i and also in degree distribution p_k. Left network have more nodes with same degree than the right network, where there are much more nodes with lower degree and a few nodes with very high degree (this is more likely for real networks). Because of that the degree distribution p_k of left network has smaller differences between the nodes.

1.2 Connected components

Given criterion:

$$n - c \leq m \leq \binom{n-c+1}{2}$$

Proof. First we are going to prove the left inequality with math induction.

Base step

Let's assume that n = 1, where n is number of nodes in the graph. Because we have only one node there are no links (m = 0) and only one connected component (c = 1).

$$n - c \leq m$$
$$1 - 1 \leq 0$$
$$0 \leq 0$$

Induction step

Next we assume that this inequality is true for n. We try to prove that it is true for n + 1.

$$(n+1) - c \leq m$$
$$n + 1 - c \leq m$$

We have two different options. We can have one large connected component (c = 1). In this case there are at least n links between the nodes (we have n + 1 nodes, because we try to prove for that this inequality is true for n + 1), because if we have one connected component all nodes must be connected (m = [n, ∞]).

$$n + 1 - c \leq m$$
$$n + 1 - 1 \leq m$$
$$n \leq m$$
$$n \leq n$$

If there are no links in the graph (m = 0) we have n + 1 connected components (c = n + 1).

$$n + 1 - c \leq m$$

$$n + 1 - n - 1 \leq m$$
$$0 \leq m$$
$$0 \leq 0$$

□

Now we have to prove the right side of the inequality:

$$m \leq \binom{n - c + 1}{2}$$

Proof. **Base step**

Let's assume that n = 1, where n is number of nodes in the graph. Because we have only one node there are no links between them (m = 0) and only one connected component (c = 1).

$$m \leq \binom{n - c + 1}{2}$$
$$0 \leq \binom{1 - 1 + 1}{2}$$
$$0 \leq 0$$

Induction step

Next we assume that this inequality is true for n. We try to prove that it is true for n + 1.

$$m \leq \binom{n + 1 - c + 1}{2}$$
$$m \leq \binom{n - c + 2}{2}$$

Again we have two different options. We can have one large connected component (c = 1). In this case there are at least n links between the nodes (we have n + 1 nodes, because we try to prove for that this inequality is true for n + 1), because if we have one connected component all nodes must be connected (m = [n, ∞]).

$$m \leq \binom{n - 1 + 2}{2}$$
$$n \leq \binom{n + 1}{2}$$
$$n \leq \frac{(n + 1)! \cdot 2!}{(n - 1)!}$$
$$n \leq \frac{n \cdot (n + 1) \cdot 2}{1}$$

If there are no links in the graph (m = 0) we have n + 1 connected components (c=n + 1).

$$m \leq \binom{n - n - 1 + 2}{2}$$
$$0 \leq \binom{1}{2}$$

$$0 \leq 0$$

□

If we want to have a connected graph, we know that there must be one connected component. From the above inequality we know that if we have only one connected component then the maximum m is $\binom{n}{2}$. So the criterion for m is:

$$m \geq \binom{n}{2}$$

We can use this criterion to compute the size of the largest weakly connected component, because if we see that we have more than $\binom{n}{2}$ links between nodes we know that there is one large connected component, which includes all nodes.

1.3 Weak and strong connectivity

If the algorithm from the labs follow the links in any direction (undirected graph) it will find all the weakly connected components.

If the algorithm follow the links in the proper direction we will find a strongly connected components. The algorithm will find all the nodes that we can reach from the starting node (some of that nodes does not belong to the strongest connected component because we can not reach the starting node).

And if the algorithm follow the links in the opposite direction it will find more connected components that it should found. Also we could not reach the starting point.

I did not implement algorithm, I only use networkx library to calculate strongest connected components and the size of the largest one.

1

```
1
2  import networkx as nx
3
4  g = nx.DiGraph()
5  for i in range(5):
6      g.add_node(i)
7  with open("data/enron") as f:
8      while f.readline().startswith("#"):
9          pass
10     row = f.readline().split(" ")
11     while row[0] != '':
12         g.add_edge(int(row[0]), int(row[1]))
13         row = f.readline().split(" ")
14 print("finish")
15
16 SCC = max(nx.strongly_connected_components(g), key=len)
17 number = nx.number_strongly_connected_components(g)
18 print(f"Number of strongly connected component: {number}")
19 print(f"Size of the largest strongly connected component: {len(SCC)}")
```

Number of strongly connected components in Enron e-mail communication network is 78058 and size of the largest connected component is 9164. The number of all nodes in this network is 87273 and we get that in the largest connected component there are only 9164 of the nodes. This is a little bit surprisingly since most of the nodes are not in the largest connected component (there are only 9164 out of 87273 nodes in the strongest connected component). That means that there are a lot of emails do not get response. Also we can see that number of all connected

components is 78058 which is almost the same as the number of all nodes 87273. That means that in our network there are very small groups of people that sends email to each other.

1.4 Node network clustering

In a double star network there are two nodes that are connected to each other and to all other nodes, but there must be no other links except those. We call these two nodes node 1 and node 2. These two nodes have degree $n-1$ and number of triangles equal to $n-2$. First we prove that the average clustering coefficient is $\langle C \rangle \to 1$, when $N \to \infty$.

Proof. Node 1 and 2 have degree $k_i = N-1$ and number of triangles $t_i = N-2$, so the local clustering coefficient is:

$$C_i = \frac{2(N-2)}{(N-1)(N-2)}$$

$$C_i = \frac{2}{N-1}$$

All other nodes have degree $k_i = 2$ and number of triangles $t_i = 1$. The local clustering coefficient for these nodes is:

$$C_i = \frac{2 \cdot 1}{2 \cdot 1}$$

$$C_i = 1$$

From these two equations we can calculate the **average clustering coefficient**:

$$\langle C \rangle = \frac{1}{N} \sum_{i=1}^{N} C_i$$

$$\langle C \rangle = \frac{1}{N} \cdot ((N-2) \cdot 1 + 2 \cdot \frac{2}{N-1})$$

$$\langle C \rangle = \frac{1}{N} \cdot (N - 2 + \frac{4}{N-1})$$

$$\langle C \rangle = 1 - \frac{2}{N} + \frac{4}{N \cdot (N-1)}$$

If $N \to \infty$ then $\langle C \rangle \to 1$.

□

Now we prove that **global clustering coefficient** is $C \to 0$, when $N \to \infty$.

Proof. In double star network there are $N-2$ triangles and $N^2 - 2N$ triples. When we insert these two values in the equation for calculation global clustering coefficient we get:

$$C = \frac{3 \cdot (N-2)}{N^2 - 2N}$$

$$C = \frac{3 \cdot (N-2)}{N \cdot (N-2)}$$

$$C = \frac{3}{N}$$

When $N \to \infty$ then $C \to 0$.

□

Another example of network where $\langle C \rangle \to 1$ and $C \to 0$, when $N \to \infty$ is triple star network. In the triple star network we have three main nodes that are connected to each other and they are also connected to all other nodes (but there are no other links between these other nodes).

So if we have three main nodes, their degree will be N - 1 and number of triangles N - 2. From there we get that the local clustering coefficient is $C_i = \frac{2 \cdot (N-2)}{(N-1) \cdot (N-2)}$. For all other nodes we get that they have degree of 3 and number of triangles 3, so the local clustering coefficient is $C_i = \frac{2 \cdot 3}{3 \cdot 2}$ which gives us 1.

Now we can calculate average clustering coefficient:

$$\langle C \rangle = \frac{1}{N} \sum_{i=1}^{N} C_i$$

$$\langle C \rangle = \frac{1}{N} \cdot ((N-3) \cdot 1 + 3 \cdot \frac{2}{N-1})$$

$$\langle C \rangle = \frac{1}{N} \cdot (N - 3 + \frac{3}{N-1})$$

$$\langle C \rangle = 1 - \frac{3}{N} + \frac{3}{N \cdot (N-1)}$$

When $N \to \infty$ then $\langle C \rangle \to 1$.

We can also see that the global clustering coefficient goes to when N goes to ∞.

Another example is graph shown in picture below:

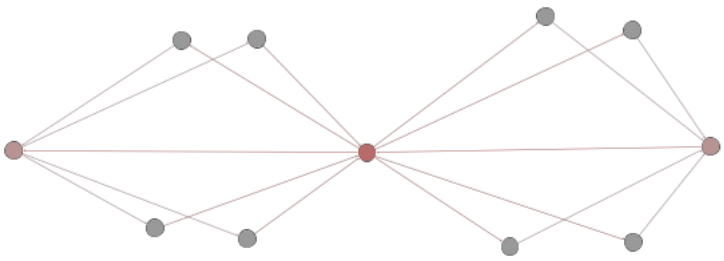

Figure 1: Example of network.

1.5 Effective diameter evolution

In this exercise I did not implement algorithm for computing effective diameter evolution. I only calculate number of nodes and average degree which are shown in Table 1.

Code for calculating number of nodes and average degree for all three given networks.

Graph	Number of nodes	Average degree
Aps 2010-2011	18985	4.369
Aps 2010-2012	38356	5.73
Aps 2010-2013	56473	7.095

Table 1: Number of nodes and average degree for all three networks.

```
    1
1
2 graphs = ["aps_2010_2011", "aps_2010_2012", "aps_2010_2013"]
3     nodes = [18985, 38356, 56473]
4
5     """ First graph - 2010_2011 """
6     for j in range(len(graphs)):
7         g = nx.Graph()
8         for i in range(1, nodes[j] + 1):
9             g.add_node(int(i))
10        with open(f"data/{graphs[j]}") as f:
11            while f.readline().startswith("#"):
12                pass
13            row = f.readline().split(" ")
14            while row[0] != '':
15                g.add_edge(int(row[0]), int(row[1]))
16                row = f.readline().split(" ")
17
18        print(f"Number of nodes in graph 2010_2011: {g.number_of_nodes()}")
19        degrees = [d for (n,d) in g.degree]
20        print(f"Average degree is: {round(sum(degrees) / g.number_of_nodes(),
    3)}")
```

2 Graph models (3 points)

2.1 Random node selection

If we want that our algorithm runs in constant time O(1), where node i will be selected with probability $\frac{k_i}{2m}$ (k_i is its degree and m number of links) we take edge list as network representation. In that way every node i will be selected with the right probability, which is $\frac{k_i}{2m}$ and this will run in constant time O(1). So nodes with higher degree will be selected with higher probability, because there will be more edges with starting point of these node. (For example if we have edge list like this: AB, BA, BC, CB node B will be selected with highest probability $\frac{2}{4}$ because it has highest degree.)

Algorithm 1 Random node selection (Edge_list)

1: i $\leftarrow$ select random number of list *Edge_list*
2: **return** *Edge_list*[*i*]

2.2 Node linking probability

We want to show that the expected node degree k_i is proportional to some non-negative number v_i. We know that the probability p_{ij} is proportional to $v_i \cdot v_j$, so we can write $p_{ij} = \alpha_{ij} \cdot v_i \cdot v_j$,

where α_{ij} is some constant. For each node it is true that node degree is equal to sum of all probabilities:

$$k_i = \sum_j \alpha_{ij} \cdot v_i \cdot v_j$$

We can put v_i out of the sum, because the sum is only dependent on the parameter j.

$$k_i = v_i \cdot \sum_j \alpha_{ij} \cdot v_j$$

Now we see that k_i is proportional to v_i, because k_i is written as product of v_i and something else. If we expose v_i from this equation we can insert this into equation for p_{ij} and we get:

$$p_{ij} = \frac{k_i}{\sum_j \alpha_{ij} \cdot v_j} \cdot \frac{k_j}{\sum_i \alpha_{ij} \cdot v_i}$$

From this equation we can see that if the values of v_i and v_j are smaller than 1, then the probability will be higher, because we will have smaller value in the denominator than in nominator. But if these two values will be higher than 1, then the denominator will increase faster than the nominator, so the probability of the link will be smaller.

2.3 Node degree distributions

Figure 2 shows us double logarithmic curves of all four degree distributions.

For the real Facebook network, we can see that the curve of degree distribution follows a power law. With higher degree the degree distributions is lower. That kind of network is called scale free network.

From the Erdos-Renyi graph's degree distribution we can see that it creates hubs around the average degree.

By plotting degree distribution using binomial distribution $p_k \cong \frac{\langle k \rangle^k \cdot e^{-\langle k \rangle}}{k!}$ we can see that we get almost the same degree distribution curve as with the Erdos-Renyi graph.

At the end we construct Barabasi Albert model with our preferential attachment model implementation which degree distribution is very similar to degree distribution of Facebook network. This netwrok is also scale free, because the degree distribution decreases when the number of degrees increases.

Preferential attachment model:

1

```
1
2  def construct_barabasi_albert_graph(average, n):
3      g = nx.Graph()
4
5      start_nodes = int(np.ceil(average)) + 1
6      g = nx.complete_graph(start_nodes)
7
8      # number of new selected nodes
9      new_links = int(np.ceil(average / 2))
10
11     # create list of nodes, node will be selected with probability to their
       degree and links to them
12     prob = []
13     degrees = [d for (n, d) in g.degree()]
14     for i in g.nodes():
```

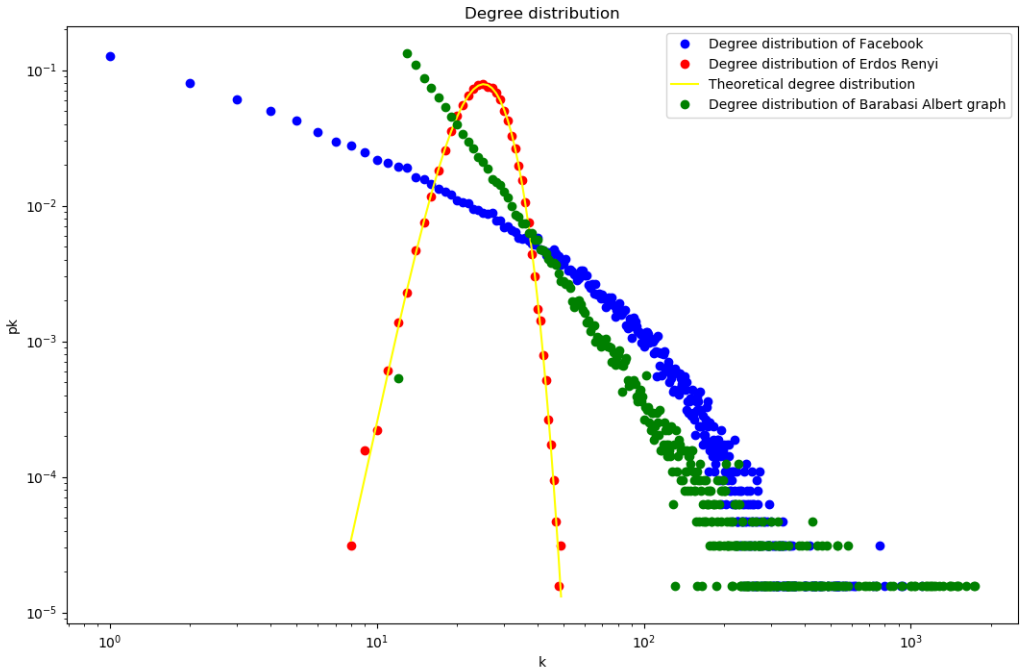

Figure 2: Degree distributions.

```
        for j in range(degrees[i]):
            prob.append(i)

    # create a list with all new nodes that will be added to graph
    new_nodes = [i + start_nodes for i in range(n - start_nodes)]
    for i in new_nodes:
        g.add_node(i)
        for j in range(new_links):
            new_node2 = prob[random.randint(0, len(prob) - 1)]
            g.add_edge(i, new_node2)
            prob += [i, new_node2]

    return g
```

The code for this exercise:

1

```

import collections
import random

import matplotlib.pyplot as plt
import networkx as nx
import math
import numpy as np


def degree_distribution(G):

```

```
    # degree distribution pk
    degree = [v for (n,v) in G.degree]
    counter = collections.Counter(degree)
    n = G.number_of_nodes()
    pk = [(d, nk / n) for (d, nk) in counter.items()]

    return pk


def average_degree(G):

    degree = [v for (n,v) in G.degree]
    average_degree = sum(degree) / G.number_of_nodes()
    return average_degree


def plot_pk(pk, style, color):

    k = [d for (d, n) in pk]
    nk = [n for (d, n) in pk]

    plt.loglog(k, nk, style, color=color)


def construct_erdos_renyi(G):

    n = G.number_of_nodes()
    m = G.number_of_edges()
    random_G = nx.gnm_random_graph(n,m)
    return random_G


def theoretical_degree_distribution(random_G):
    average = average_degree(random_G)
    degree = [v for (n, v) in random_G.degree]
    counter = collections.Counter(degree)
    pk = [(k, (math.pow(average, k) * math.exp(-average)) / (math.factorial(k))
    ) for (k, n) in counter.items()]
    return pk


def construct_random_graph(G):

    average = average_degree(G)
    n = G.number_of_nodes()
    m = np.ceil(average / 2)

    G = nx.barabasi_albert_graph(n, int(m))

    return G


def construct_barabasi_albert_graph(average, n):
    g = nx.Graph()

    start_nodes = int(np.ceil(average)) + 1
    g = nx.complete_graph(start_nodes)

    # number of new selected nodes
```

```
71      new_links = int(np.ceil(average / 2))
72
73      # create list of nodes, node will be selected with probability to their
        degree and links to them
74      prob = []
75      degrees = [d for (n, d) in g.degree()]
76      for i in g.nodes():
77          for j in range(degrees[i]):
78              prob.append(i)
79
80      # create a list with all new nodes that will be added to graph
81      new_nodes = [i + start_nodes for i in range(n - start_nodes)]
82      for i in new_nodes:
83          g.add_node(i)
84          for j in range(new_links):
85              new_node2 = prob[random.randint(0, len(prob) - 1)]
86              g.add_edge(i, new_node2)
87              prob += [i, new_node2]
88
89      return g
90
91
92  if __name__ == "__main__":
93
94      G = nx.read_adjlist("data/facebook")
95      G = nx.Graph(G)
96
97      plt.xlabel("k")
98      plt.ylabel("pk")
99      plt.title("Degree distribution")
100
101     pk = degree_distribution(G)
102     plot_pk(pk, 'o', 'blue')
103
104     random_G = construct_erdos_renyi(G)
105     pk_random = degree_distribution(random_G)
106
107     plot_pk(pk_random, 'o', 'red')
108
109     pk_theoretical = theoretical_degree_distribution(random_G)
110     pk_theoretical = sorted(pk_theoretical, key=lambda x: x[0])
111     plot_pk(pk_theoretical, '-', 'yellow')
112
113     pk_barabas = degree_distribution(construct_barabasi_albert_graph(
        average_degree(G), G.number_of_nodes()))
114     plot_pk(pk_barabas, 'o', 'green')
115
116     plt.legend(["Degree distribution of Facebook", "Degree distribution of
        Erdos Renyi",
117                 "Theoretical degree distribution", "Degree distribution of
        Barabasi Albert graph"],
118                loc='upper right')
119     plt.show()
```

3 Node position (1.5 points)

Firstly we will compute all three scores and then we will use Sperman's correlation coefficient, to see which of these three scores is the best to predict loads.

First measure we calculate is **node degree** k_i and it gave us a correlation of 0.279 (calculated with Sperman).

The **node's clustering coefficients** $C_i = \frac{2t_i}{k_i(k_i-1}$ calculates the ratio between links existing between node's neighbors and the maximum possible number of links that could exist between all the neighbors. Because we can see that none of the neighbors of some node in given network are connected we can see that the score for the node's clustering coefficient will be 0 (t_i, which means number of triangles will be zero). That means we can not use this measure to predict loads.

At the end we calculate the **node's harmonic mean distances** $\ell_i^{-1} = \frac{1}{n-1}\sum_j \frac{1}{d_{ij}}$, which represent the closeness of locations to all other locations. This measure gives us a correlation of 0.633 (calculated with Sperman), which is much better than a correlation of node degree. This is expected because we can see that in the given network there are some nodes that are between many other nodes (like Zadobrova, Koseze, Kozarje and so on). Because of the highest score we will use this measure to predict loads.

From results we can see that nodes with the highest scoring are in the center of the Slovenia highway (Table 2).

NODE NAME	HARMONIC MEAN DISTANCE	TRAFFIC LOADS
Kozarje	0.123	35759.156
Koseze	0.122	50232.028
Zadobrova	0.12	33779.16
Malence	0.12	55297.865
Brdo	0.118	35759.156
Slivnica	0.115	20031.146
Vič	0.114	20686.532
Celovška	0.113	28813.97
Zaloška	0.113	20686.532
Litijska	0.113	31067.82

Table 2: Top ten locations according to the best node measure.

The python code for this exercise:

```
1
```

```
1
2 import networkx as nx
3 from scipy.stats import spearmanr
4 from bcolors import BLUE, ENDC
5
6
7 def read_values():
8
9     values = {}
10    names = {}
11    with open("data/highways_values") as f:
12        row = f.readline()
13        while row:
```

```
            row = row.split(" ")
            values[int(row[1])] = float(row[len(row) - 1][:-1])
            names[int(row[1])] = row[2]
            row = f.readline()

    return values, names


def harmonic_mean(G):

    h = []

    for node in G.nodes():
        shortest_path = nx.single_source_shortest_path_length(G, node)
        s = 0
        for v in shortest_path.values():
            if v != 0:
                s += 1/v
        l_node = (1 / (len(shortest_path) - 1)) * (s)
        h.append((node, l_node))

    return h


def node_measures(G):

    """ Node degree """
    print(f"{BLUE} Node degree {ENDC}")
    node_degree = [degree for (node, degree) in G.degree]
    print(node_degree)

    """ Node clusteirng coefficient """
    # C = 2*t / (k * (k-1))
    print(f"{BLUE} Clustering coefficient {ENDC}")
    clustering_coefficient = [v for k, v in nx.clustering(G).items()]
    print(clustering_coefficient)

    """ Closeness centrality """
    print(f"{BLUE} Closeness centrality {ENDC}")
    closeness_centrality = [v for (k, v) in harmonic_mean(G)]
    print(closeness_centrality)

    values, names = read_values()
    val = [v for k, v in values.items()]

    """ Spearman """
    Scorr = spearmanr(node_degree, val)
    print(f"Spearman value: {Scorr[0]}")

    Scorr = spearmanr(closeness_centrality, val)
    print(f"Sperman value: {Scorr[0]}")


def traffic_loads():

    values, names = read_values()

    hm = harmonic_mean(G)
    hm.sort(key=lambda x: x[1], reverse=True)
```

```
73
74      for i in range(10):
75          node = int(hm[i][0])
76          print(f"{i+1}. Name: {names[node]}, harmonic distance: {round(hm[i][1],
        3)}, traffic load: {round(values[node], 3)}")
77
78
79  if __name__ == "__main__":
80
81      G = nx.read_adjlist("data/highways")
82      G = nx.Graph(G)
83
84      node_measures(G)
85      traffic_loads()
```