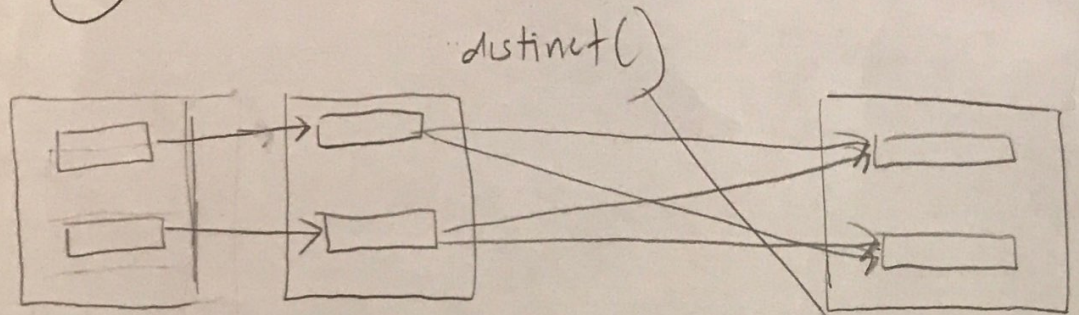
Question 1

(2)



map(ParserC(). parsestr)

map(...x.get(trim), split_receivers...--)

flatMap(...val-by-vec,...)

filter(lambda x: ... valid_mail -- enrun_check)...--)

distinct

Part (3)

Narrow Dependencies:
{map, map, flatmap, filter}

(4)

Wide Dependency:
{Distinct}

CS5234: Project PDF

Question 1.1.

Include the Python code for your chosen function, and briefly describe its implementation highlighting any challenges and pitfalls you had to deal with. (20%)

Question 1.1 Code

In [ ]:

```
# Helper function to obtain the tuples of the desired format
# This function essentially takes the list [x.get('To'), x.get('Cc'), x.get('Bcc')]
# email of irrelevant punctuation, and then breaks the strings into invidivual emai
# these smaller strings back into a list.
def split_receivers(lst):
        if (type(lst[0]) == str):
            toos = re.sub("\s", "", lst[0])
            toos = toos.split(",")
        else:
            toos = []
        if (type(lst[1]) == str):
            ccs = re.sub("\s", "", lst[0])
            ccs = ccs.split(",")
        else:
            ccs = []
        if (type(lst[2]) == str):
            bccs = re.sub("\s", "", lst[0])
            bccs = bccs.split(",")
        else:
            bccs = []
            receivers =  toos + ccs + bccs
            return receivers

## Question 1 Code for selected function)
def extract_email_network(rdd):
    rdd_mail = rdd.map(Parser().parsestr)

    val_by_vec = lambda x: [(yield(x[0], x[1][i], x[2])) for i in range(len(x[1]))]

    # Getting the desired tuples in the specified format with from address, (to, cc
    rdd_full_email_tuples = rdd_mail.map(lambda x: (x.get('From'), split_receivers(
    rdd_email_triples = rdd_full_email_tuples.flatMap(val_by_vec)

    # Regex Pattern for Identifying Legitimate Emails
    email_regex = "[!#$%&'*+-/=?^_`{|}~\w]+@(\w+\.\w+)*\.{0,1}\w*[a-zA-Z]"
```

```
    valid_email = lambda s: True if re.compile(email_regex).fullmatch(s) else False

    # Checking if it is indeed an enron email
    enron_check = lambda s: True if re.compile('.+enron.com').fullmatch(s) else Fals

    # Applying the filter operation for the two steps described immediately above
    rdd_email_triples_enron = rdd_email_triples.filter(lambda x: ( x[0] != x[1])
                                                       and (valid_email(x[0]) and va
                                                     \  and (enron_check(x[0]) and en

    # returning the distinct emails out the rdd output
    return rdd_email_triples_enron.distinct()
```

In [6]:

```
## The code for the output printed below
rdd_mail.map(lambda x: (x.get('From'), [x.get('To'), x.get('Cc'),  x.get('Bcc')], x.

print(('mark.rodriguez@enron.com',
 ['george.mcclellan@enron.com, daniel.reck@enron.com, stuart.staley@enron.com, \n\tr
  None,
  None],
 'Mon, 28 Aug 2000 06:57:00 -0700 (PDT)'))
```

```
('mark.rodriguez@enron.com', ['george.mcclellan@enron.com, daniel.reck
@enron.com, stuart.staley@enron.com, \n\tmichael.beyer@enron.com, kevi
n.mcgowan@enron.com, \n\tjeffrey.shankman@enron.com, mike.mcconnell@en
ron.com, \n\tpaula.harris@enron.com', None, None], 'Mon, 28 Aug 2000 0
6:57:00 -0700 (PDT)')
```

Challenges and Pitfalls

Above, you can see the of printed output of the code used to extract the relevant entries from the email file. The goal of this function is to output pairwaise the sender, receiver, and timestamp in the following format: (sender, receiver, timestamp).

First Tricky Part

The tricky part is that that many of the emails have formatting syntax such as "\n\t" that must be stripped from the text.

Secondly, in order to properly fromat these emails, you must recognize that each individual email is not necessarily its own entity. For instance, the following obtained from the [x.get('To'), x.get('Cc'), x.get('Bcc')]:

['george.mcclellan@enron.com (mailto:george.mcclellan@enron.com), daniel.reck@enron.com (mailto:daniel.reck@enron.com), stuart.staley@enron.com (mailto:stuart.staley@enron.com), \n\tmichael.beyer@enron.com (mailto:tmichael.beyer@enron.com), kevin.mcgowan@enron.com (mailto:kevin.mcgowan@enron.com), \n\tjeffrey.shankman@enron.com (mailto:tjeffrey.shankman@enron.com), mike.mcconnell@enron.com (mailto:mike.mcconnell@enron.com), \n\tpaula.harris@enron.com (mailto:tpaula.harris@enron.com)', None, None]

is one long string. It is misleading because one might initially think that "daniel.reck@enron.com (mailto:daniel.reck@enron.com)" is an a string by itself. This is NOT the case. This fact was one of the challenges of question 1. I did not instantly recognized that the entire sequence is a string. Instead, I initially thought that each individual email was already a string by itself. Only after I properly recognized that the whole thing is string could I successfully strip all the formatting syntax as well as split the emails into respective tuples.

Second Tricky Part

Another tricky part of the output (in blue above) are the "None." The "None" refers to the fact that some emails do not have an people "Cced, Bcced, or even To". None values are not relevant to the final output of the function and hence, they are essentially ignored in the final output, meaning, for instance, there is not output for: ('george.mcclellan@enron.com (mailto:george.mcclellan@enron.com)", None, Timestamp). We programmed the code in a way such that these occurences are taken into account but will not cause our code to error in transformations performed later on.

Hence, the function "split_receivers" takes into account these various things. It outputs the tuple: (Sender, [Receivers], Timestamp) such that "Nones" are essentially removed and each email is now a string by itself. After this, it feeds it into the next step ("val_by_vec" function) which displays every pairwise (sender, receiver, and timestamp).

Question 2.1

Use the code you developed for Questions 3 or 4 of the final project to compute the total weighted degree of all edges originating at (respectively attracted by) the 20% highest out-degree (respectively, in-degree) nodes in the network slice you selected. Briefly describe the methodology you used and your findings. Does the 80/20 rule indeed apply to your chosen network slice?

We first convert the rdd to a weighted network that only extracts the emails sent between March 1, 2000 to March 1, 2001.

In [ ]:

```
## Collecting the relevant slices of data
rdd1 = convert_to_weighted_network(rdd_for_degrees, drange = (datetime(2000,3,1,tzi
```

Methodology

a) We then obtain the out_degree and in_degree functions from the "get_out_degree" and "get_in_degree" functions.

b) After, we take the top 2159 nodes because these constitute approximately the top 20%.

c) We simply take the sum of degrees for the nodes (for in and out degrees respectively) and divide by the total number of degrees (for in and out degrees respectively).

In [112]:

```
from datetime import datetime, timezone

## Collecting the relevant slices of data
rdd1 = convert_to_weighted_network(rdd_for_degrees, drange = (datetime(2000,3,1,tzi

# Computing the out-degrees
out_degrees = get_out_degrees(rdd1)

# Since there are ~ > 10000 nodes in this problem, taking the first 2159 is approxi
top_20 = sc.parallelize(out_degrees.collect()[0:2159]).map(lambda x:x[0]).sum()

# The total number of originating edges
all_nodes = out_degrees.map(lambda x: x[0]).sum()

# Obtaining the what percent of all nodes originate from the top 20%
print(top_20/all_nodes)
print("As you can see the final value is NOT close to 80% at approximately 99.5%")
```

The final value of the top 20% is NOT close to 80% and is at approximately 99.5 for out-degree weights%

In [114]:

```
# In-Degrees
in_degrees = get_in_degrees(rdd1)#.collect()

# Since there are ~ > 10000 nodes in this problem, taking the first 2159 is approxi
top_20_in = sc.parallelize(in_degrees.collect()[0:2159]).map(lambda x: x[0]).sum()

# The total number of entering edges
all_nodes_in = in_degrees.map(lambda x: x[0]).sum()

# Obtaining the what percent of all nodes are entering the top 20%
print(top_20_in/all_nodes)
print("As you can see the final value is close to 80% at approximately 86.0%")
```

The final value of the top 20% is 86%, relatively close to 87% for in-degree weights%

Question 2.2)

Use the functions you implemented for Questions 2 and 3 of the final project to compute kmax (for both in and out degrees) and the number of nodes for the sub-slices containing the Emails sent within the first n months, where 1 ≤ n ≤ 12, of your chosen network slice. Briefly describe the methodology you used and your findings. Does kmax for either in or out degrees (or both) indeed grow linearly with the number of nodes in your chosen network slice?

Description for Q2.2 Code

Essentially a loop is performed to extract the time slices ranging from 1 months to a time slice that takes up 12 months. After the relevant rdd's are obtained for these time slices, we simply call max on the rdd's to obtain a list that records each time period's respective max degrees.

In [ ]:

```
## Question 2.2 Code)
#Getting the rdd for various n
out_degree_k = []
in_degree_k = []

n = list(range(1,13))
for i in [4,5,6,7,8,9,10,11,12,13,14, 15]:
    if i <= 12:
        k = convert_to_weighted_network(combined_parsed, drange = (datetime(2000,3,1
    else:
        k = convert_to_weighted_network(combined_parsed, drange = (datetime(2000,3,1

    out_degree_k.append(get_out_degrees(k).map(lambda x: x[0]).max())

    in_degree_k.append(get_in_degrees(k).map(lambda x: x[0]).max())
```

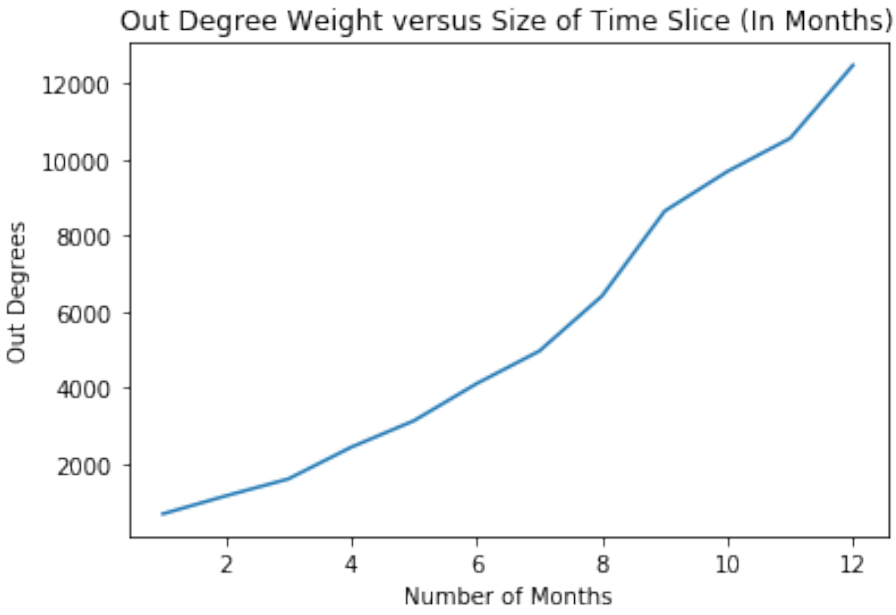
In [109]:

```
## The outputs for the code above)

out_degree_k = [684, 1155, 1602, 2435, 3131, 4106, 4963, 6412, 8644, 9686, 10557, 12

in_degree_k = [227, 388, 600, 890, 1107, 1395, 1718, 2123, 2475, 2678, 2775, 2997]

n = [1,2,3,4,5,6,7,8,9,10,11,12]

import matplotlib.pyplot as plt

display(plt.plot(n, out_degree_k))
plt.title('Out Degree Weight versus Size of Time Slice (In Months)')
plt.xlabel('Number of Months')
plt.ylabel('Out Degrees')
```

```
[<matplotlib.lines.Line2D at 0x122e13940>]
```

Out[109]:

```
Text(0, 0.5, 'Out Degrees')
```



Description of Out-Degree vs Months Graph Above

Indeed, if we look at the out degree weight versus the months, there is definitely a linear, almost slightly exponential trend.

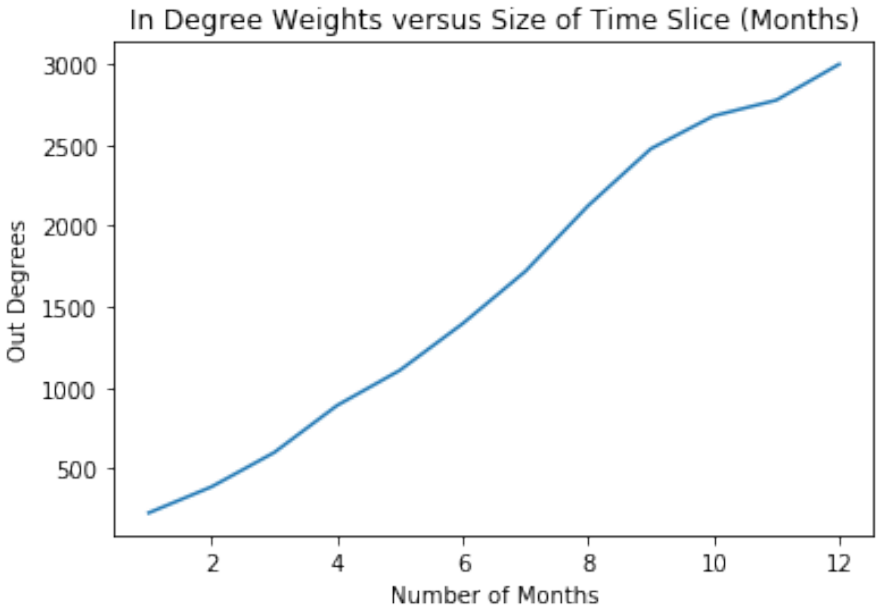
In [111]:

```
display(plt.plot(n, in_degree_k))
plt.title('In Degree Weights versus Size of Time Slice (Months)')
plt.xlabel('Number of Months')
plt.ylabel('Out Degrees')
```

```
[<matplotlib.lines.Line2D at 0x122fb52b0>]
```

Out[111]:

```
Text(0, 0.5, 'Out Degrees')
```



Description of In-Degree Weights vs Months Graph Above

Indeed, if we look at the out degrees versus the months, there is also another linear-looking trend. However, while the in-degree weight does indeed increase with months, the same trend also displays that this rate of increase is decreasing. Compared to the out-degree chart which actually displays an increasing rate of increase with the number of months, this is a notable discrepancy between the in-degree and out-degree graphs.

Question 2.3)

Use the functions you implemented for Question 4 of the final project to compute the in and out degree distributions for your chosen network slice. Graph the distributions you obtained on a log-log scale, and determine whether a straight line can be fit through the points. Briefly describe the methodology you used and your findings. Discuss whether your chosen network slice is indeed scale-free in terms of either its in or out degrees (or both), and if so, give the value of the power law exponent α (or its range) for the relevant degree distribution.

Methodology

To obtain the counts of degree weights, simply call the "get_out_degree" function and "get_in_degree_dist" function below

In [33]:

```
def get_out_degree_dist(rdd):
    return get_out_degrees(rdd).map(lambda x: (x[0], 1)).reduceByKey(lambda x,y: x+

def get_in_degree_dist(rdd):
    return get_in_degrees(rdd).map(lambda x: (x[0], 1)).reduceByKey(lambda x,y: x+ y

power_out_degrees = get_out_degree_dist((rdd1))
power_in_degrees = get_in_degree_dist((rdd1))
```

Code for Calculating the Counts of Degree Weights

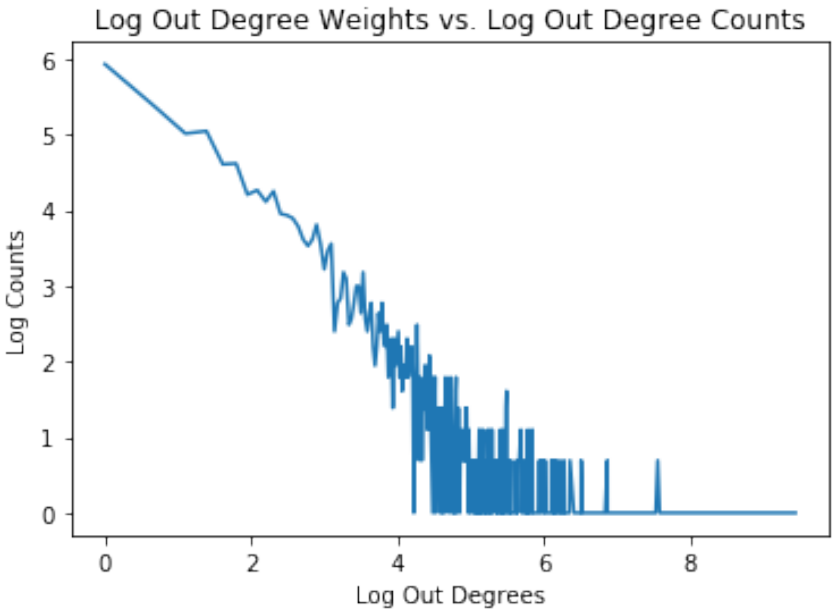
In [106]:

```
import numpy as np
in_n = np.array([i[0] for i in in_degree_hist])
out_n = np.array([i[0] for i in out_degree_hist])
in_degrees = np.array([i[1] for i in in_degree_hist])
out_degrees = np.array([i[1] for i in out_degree_hist])

plt.plot(np.log(out_n), np.log(out_degrees))
plt.xlabel('Log Out Degrees')
plt.ylabel('Log Counts')
plt.title('Log Out Degree Weights vs. Log Out Degree Counts')
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7: Runti
meWarning: divide by zero encountered in log
  import sys
```

Out[106]:

```
Text(0.5, 1.0, 'Log Out Degree Weights vs. Log Out Degree Counts')
```



Log Out-Degree Weights vs Log Out-Degree Count

Above, we do see a somewhat though not completely linear downward trend. However, what is interesting to note is there is dramatically more fluctuation as the size of the out-degree increases.

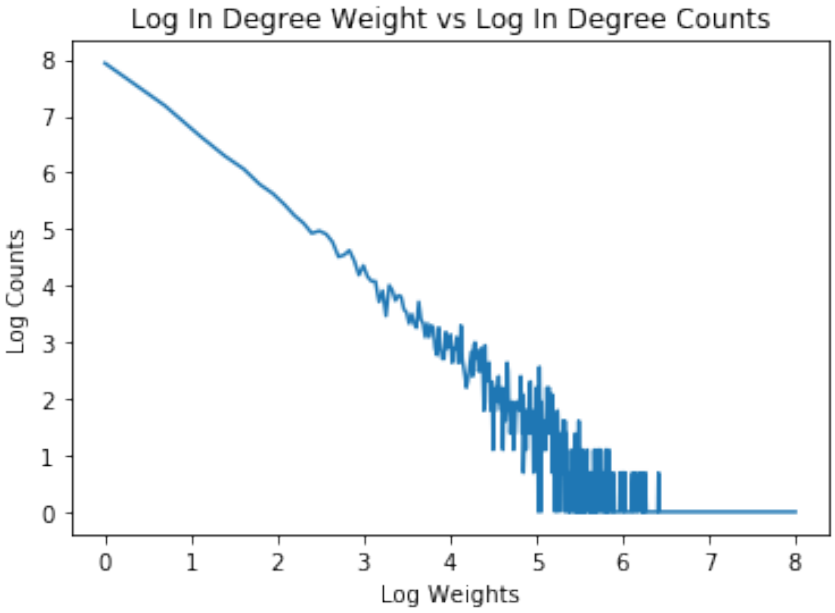
In [105]:

```
plt.plot(np.log(in_n), np.log(in_degrees))
plt.title('Log In Degree Weight vs Log In Degree Counts')
plt.xlabel('Log Weights')
plt.ylabel('Log Counts')
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: Runti
meWarning: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```

Out[105]:

```
Text(0, 0.5, 'Log Counts')
```



Log-In Degree Weights vs Log In-Degree Counts

Above, we also see a somewhat though not concrete linear downward trend with increasing fluctuation of the points with in-degree size. However, this fluctuation tends to be much smaller than that displayed by the out-degree graph preceding.

A-Value (Power) Methodology:

Computing the a-value for each individual degree weight (for in-degrees and out-degrees) and then averaging over all of the a-values. This value obtained should approximate the slope of the line that can be fit through the in-degree and out-degree line graphs.

In order to calculate the a-values (the power), we must first obtain their probability distrubution, which can be found by dividing the counts for each respective out-degree and in-degree weight by the total sum of respective out-degree and in-degree counts.

In [115]:

```
# Obtaining the sum total counts of degree weights for out-degrees and in-degrees
total_out_degrees = np.sum(out_degrees)
total_in_degrees = np.sum(in_degrees)

# Obtaining the estimate of the probability density distribution by dividing the res
#in-degree weight frequencies by the sum of the total of counts for the respective d
out_pdf = out_degrees/total_out_degrees
in_pdf = in_degrees/total_in_degrees

# Solving for "a" by plugging in to the equation: p(k) = k^-a where k is the degree-
out_degree_power = -np.log(out_pdf)/np.log(out_n)
in_degree_power = -np.log(in_pdf)/np.log(in_n)
```

Conclusion: Is this really a scale-free network?

Both the in-degree and out-degree email networks both seem to be scale-free networks. There are multiple reasons for this:

1) Both exhibit linearity (the points tend to fall on a line) after we take the log-scale of the counts and degree weights.

2) Most nodes have a very small degree, but there are a few nodes that have disproportionately large degrees. These nodes are essentially "hubs" which have a magnitude of degree much larger than the other nodes. This is a common characteristic of scale-free networks.

However, there is also some evidence against the fact that our email network is a power law network.

After computing the power for each degree-weight (the a-value), we would expect these a-values for all the degree weights to be relatively close to one another so that the power law formula holds true: p(k) = k^a (where a is the same for all degree_weights). However, this is not the case. In fact, the powers for both the in-degree and out-degree weights exhibit an almost negative logarithmic sequentiallity when plotted against the degree_weights on the x-axis. If this were truly scale-free network, we would expect to see the a-values more or less resemble a flat line.

Below, you can see the negative logarithmic sequentiallity displayed by the poweres for both the out-degree and in-degree email networks.

In [96]:

```
plt.plot((out_n), out_degree_power)
plt.title('Power vs. Out-Degree Weight')
plt.xlabel('Out Degree Weights')
plt.ylabel('Power (A-Value) )')
plt.show()

plt.plot((in_n), in_degree_power)
plt.title('Power vs. In-Degree Weight')
plt.xlabel('In Degree Weights')
plt.ylabel('Power (A-Value) )')
plt.show()
```