CS5234/CS4234 – Final Project Report

Question 1 (60%):

1. Python Code:

```
def get_out_degrees(rdd):
    Q3i_RDD = rdd.map(lambda x: (x[0], x[2])) \
                 .reduceByKey(operator.add) \
                 .map(lambda x: (x[1], x[0])) \
                 .sortBy(lambda x: x[0], ascending = False)
    return Q3i_RDD
```

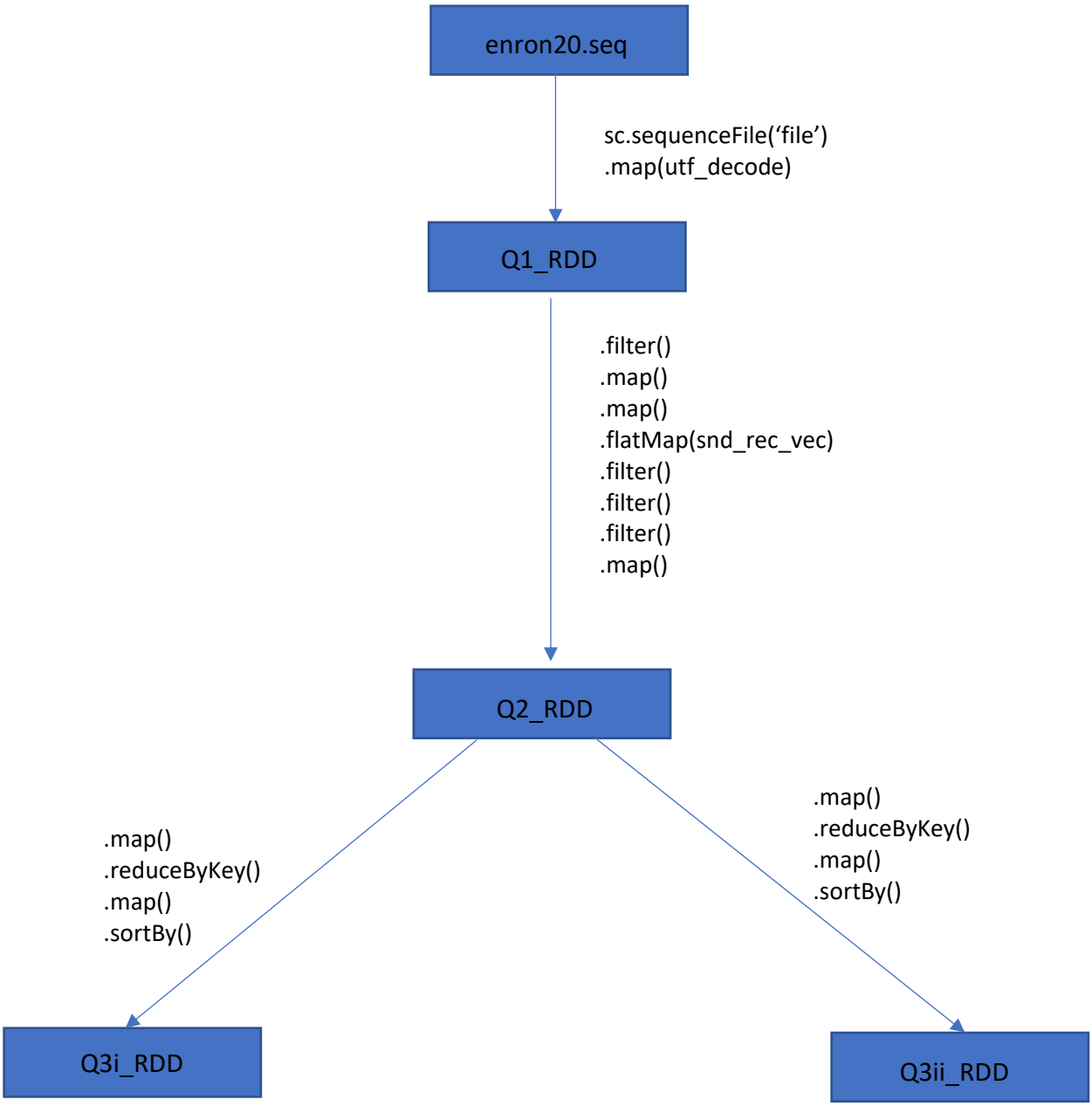
Calling function:

```
print(pretty_rdd(get_out_degrees(convert_to_weighted_network(rdd))))
```

Implementation:

- For the implementation of get_out_degrees, we get the RDD from convert_to_weighted_network function as the argument.
- In the next line we use "Q3i_RDD" as our return RDD and we get this RDD by calling rdd.map and by using a lambda expression to get the sender emails and the edges from the RDD. In the tuple pair, the senders act as Keys and the weights act as Values.
- In the third line we use "reduceByKey" and add the values which have the same set of keys to reduce the numbers of lines in the RDD.
  E.g. ('john.haggerty@enron.com', 1),
  ('john.haggerty@enron.com', 1)
  after the calling of "reduceByKey" we get
  ('john.haggerty@enron.com', 2)
- Since we need RDD pairs of (d, n) where 'd' is the weighted out-degree and 'n' being the node in the input network, we must change the returning RDD from the "reduceByKey" from (n, d) to (d, n), this is done by calling map and using a lambda expression 'x' to interchange the values of x[0], x[1].
- We need to sort the RDD in descending order of the values so the function "sortBy" is called on a lambda expression 'x' which is used to sort the RDD in terms of the value or the weighted out-degree (first value x[0] of the tuple pair).
- We use "return Q3i_RDD" which returns the output of the definition or function "get_out_degrees".
- The challenges we faced were having to reduceByKey which required the operator library to perform the addition operation. Also, Sorting the elements required a second parameter 'ascending' which stated the order of sorting. False means descending.

2. Lineage Graph:



3. Narrow Dependencies:

There are few narrow dependencies that are present in the lineage graph and an example of one narrow dependency is between "extract_email_network()" and "convert_to_weighted_network()" where the output returned by "extract_email_network()" which is "Q1_RDD" is needed by "convert_to_weighted_network()" to return the output "Q3i_RDD".

4. Wide Dependencies:

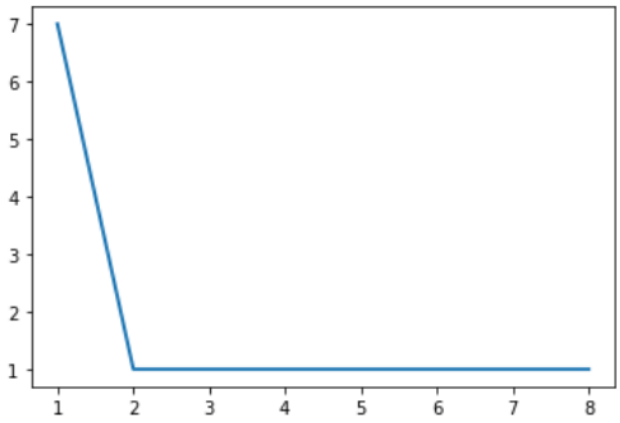
Wide Dependency in the above lineage graph is the between "convert_to_weighted_network()", "get_out_degrees()" and if we also consider the next function "get_in_degrees()" we can say that both "get_in_degrees()" and "get_out_degrees()" rely on the on the output of "convert_to_weighted_network()" that is "Q2_RDD".

Question 2 (40%):

1.

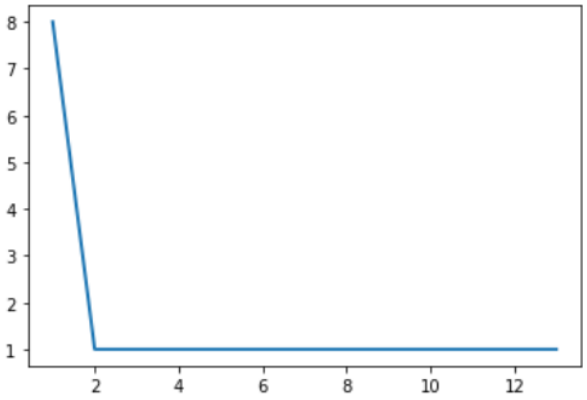
Plotting Q4_i_RDD list obtained from get_out_degree_dist(rdd) function, we have the following graph plotted using matplotlib.pyplot

```
In [40]: plt.plot(make_x(q4i), make_y(q4i), linewidth=2.0)
Out[40]: [<matplotlib.lines.Line2D at 0x227a17286d0>]
```



Similarly, we can plot the get_in_degree_dist(rdd) and obtain the following plot.

```
In [50]: plt.plot(make_x(q4ii), make_y(q4ii), linewidth=2.0)
Out[50]: [<matplotlib.lines.Line2D at 0x227a1958df0>]
```



Both the plots follow the power law graph, where 20% of the hubs have sent the most mails in get_out_degree_dist(rdd) output and 80% have sent the least mails.
Similarly, 20% of the hubs have received the most mails in get_in_degree_dist(rdd) output and 80% have received the least mails.