Question 1: Basic Network Extraction (40%)

What to do: Write a Python function `extract_email_network()` that takes an RDD `rdd` as argument and returns an RDD of triples `(S, R, T)` each of which representing an Email transmission from the *sender* `S` to the *recipient* `R` occurring at time `T`.

You can assume that `rdd` is obtained by calling the command below:

```
utf8_decode_and_filter(sc.sequenceFile('<path_to_the_input_dataset>'))
```

The function `utf8_decode_and_filter()` is available in the provided testing code (`test-driver.py`) as well as in the Lab 8 and 9 notebooks.

Each Email message has a *single* sender and *one or more* recipients. The Email address of the sender is the value of the `From` field in the message header, and the recipient Email addresses is the union of the Email addresses in the `To`, `Cc`, and `Bcc` fields. The string representing the message transmission timestamp is the value of the `Date` field. Thus, a single message may translate to multiple output triples – one for each unique pair of the message sender and one of its recipients.

The RDD returned by `extract_email_network()` must satisfy all of the following constraints:

- Every Email address appearing in either the sender or the recipient field of every output triple must be a valid Email address as per the definition in Assignments 1 and 2.
- Every Email address appearing in either the sender or the recipient field of every output triple must belong to the `enron.com` domain, i.e., the last two labels of its domain name must be `enron` followed by `com`. For example, `jane.doe@enron.com` and `joe.smart@sales.enron.com` are both valid Enron Email addresses whereas both `joe.smart@senron.com` and `joe.smart@ibm.com` are not.
- The timestamp field of every output triple must be an instance of the Python `datetime` object. Use the provided method `date_to_dt()` to convert the string timestamp in the `Date` field of the message header to an instance of `datetime` holding the equivalent time in the UTC time zone.
- All self-loops, i.e., the triples having identical sender and recipient Email addresses, must be excluded.
- All output triples must be distinct.

You can use the Python's Email parser library `email.parser` as explained in the Lab 8 and 9 notebooks to parse the Email messages and extract the values of relevant fields from their headers.

Question 2: Creating a Weighted Network (20%)

A *weighted network* is the network in which each edge is associated with a positive integer, called the edge's *weight*.

In this question, you will convert the Email network extracted in Question 1 to a weighted network in which every two nodes are connected by at most two edges (one in either direction), and the weight of each edge (a, b) is the number of Email messages sent from a to b.

What to do: Write a function `convert_to_weighted_network()` that takes one *required* argument `rdd`, which is an RDD that complies with the output format of the function `extract_email_network()` specified in Question 1, and one *optional* argument `drange`, which is a pair `(d1, d2)` of `datetime` objects with a default value of `None`. The function returns an RDD consisting of *distinct* triples `(o, d, w)` such that all of the following constraints hold:

- `(o, d, t)` is an element of the input RDD for some timestamp `t`;
- if `drange` is not `None`, then `w` is the number of edges `(o', d', t)` in the input RDD such that $(o', d') = (o, d)$ and $drange[0] \leq t \leq drange[1]$;
- if `drange` is `None`, then `w` is the number of edges `(o', d', t)` in the input RDD such that $(o', d') = (o, d)$.

Note that to make the `datetime` components of `drange` comparable to the `datetime` objects stored in the input RDD, each of them must be instantiated by providing `timezone.utc` as the value of the `tzinfo` parameter to its constructor. E.g., `datetime(2000,9,1,tzinfo=timezone.utc)` will create a `datetime` object encapsulating the time `1/9/2000 00:00 UTC`.

Question 3: Computing Basic Degree Statistics (20%)

The *weighted out-degree* (respectively, *weighted in-degree*) of a node n in a weighted network is the sum of the weights of all edges leaving (respectively, entering) n in the network.

Question 3.1 (10%)

Write a function `get_out_degrees()` that takes an RDD representing a weighted network as argument, and returns an RDD of pairs `(d, n)` satisfying the constraints below:

- `d` is a non-negative integer;
- `n` is a string holding an Email address;
- the weighted **out-degree** of `n` is `d`;
- there is *exactly one* pair `(d, n)` for each node `n` in the input network (even if its weighted out-degree is 0);
- the output is sorted in the descending lexicographical order of the integer/string pairs.

You can assume that the input RDD complies with the output format of the function `convert_to_weighted_network()` specified in Question 2.

Question 3.2 (10%)

Write a function `get_in_degrees()` that takes an RDD representing a weighted network as argument, and returns an RDD of pairs `(d, n)` satisfying the constraints below:

- `d` is a non-negative integer;
- `n` is a string holding an Email address;
- the weighted **in-degree** of `n` is `d`;
- there is *exactly one* pair `(d, n)` for each node `n` in the input network (even if its weighted in-degree is 0);
- the output is sorted in the descending lexicographical order of the integer/string pairs.

You can assume that the input RDD complies with the output format of the function `convert_to_weighted_network()` specified in Question 2.

Question 4: Computing Degree Distributions (20%)

In this question, you will write functions to generate weighted in- and out-degree distributions for a weighted network as defined in Question 3.

Question 4.1 (10%)

Write a function `get_out_degree_dist()` that takes an RDD representing a weighted network, and returns an RDD of pairs mapping each weighted out-degree of a node in the network to the number of nodes having this out-degree. The output RDD must be sorted in the *ascending* order of the out-degrees. You can assume that the input RDD complies with the output format of the function `convert_to_weighted_network()` specified in Question 2.

Question 4.2 (10%)

Write a function `get_in_degree_dist()` that takes an RDD representing a weighted network, and returns an RDD of pairs mapping each weighted in-degree of a node in the network to the number of nodes having this in-degree. The output RDD must be sorted in the *ascending* order of the in-degrees. You can assume that the input RDD complies with the output format of the function `convert_to_weighted_network()` specified in Question 2.