CS5234J – Final Project

Getting started

You will be working with a dataset containing a collection of Emails exchanged by employees of the Enron corporation in the late 1990s and early 2000s. This dataset became a public record following a high-profile investigation by the US Securities and Exchange Commission (SEC), which resulted in several Enron's top executives being prosecuted for fraud, and the company itself going bankrupt (see `https://en.wikipedia.org/wiki/Enron_scandal` for more details).

The dataset is structured as a Hadoop Sequence File comprising a collection of binary objects each of which representing a single Email message. A representative example of an Email message in the dataset can be found in the file `/home/local/ufac001/students/project2021/enron-one-sample.txt` on bigdata.

Provided datasets The following datasets are available on HDFS. Use smaller datasets for debugging, and the full dataset for completing tasks in the Project Report as necessary:

- `/user/ufac001/project2021/enron-full`: the full dataset containing approximately 0.5M Email messages.
- `/user/ufac001/project2021/samples`: a folder containing a selection of differently sized random samples of the full dataset. The size of each sample is encoded in the name of the file storing that sample.

Provided source code The following Python source code is available on Moodle:

- `project.py`: a template implementation that includes helper functions as well as place-holders for the functions you have to implement for Tasks 1 – 4;
- `test-driver.py`: a driver program that includes sample testing code along with expected outputs.

What and how to submit

1. Paste your code for the functions you have been asked to implement in Tasks 1 – 4 into the relevant place-holders in `project.py`.
2. Make sure `project.py` does not have any syntax errors, and all functions work as expected. Do not include any testing printouts.
3. Submit `project.py` on Moodle. Do not submit the testing code, or anything else that is not required.
4. You should **not** use SparkSQL or Spark DataFrames/DataSets as part of your solution (i.e. use RDDs and the RDD APIs).

Submission deadline: 10:00am, 9 August 2021

Task 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 7 and 8 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 Assignment 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 Lab 9 to parse the Email messages and extract the values of relevant fields from their headers.

Task 2: Computing Monthly Contacts (20%)

In this task, you will write a function to compute, for each sender, the month where they contacted the most people (assuming that one email address corresponds to one person).

What to do: Write a function `get_monthly_contacts()` that takes one argument `rdd`, which is an RDD that complies with the output format of the function `extract_email_network()` specified in Task 1. The function `get_monthly_contacts()` must return an RDD consisting of *distinct* triples such that, for each triple `(s, m, n)`

- there is a triple `(s, r, t)` in the input RDD for some `r` and `t`,
- `m` is a string representing a month (in the format `MM/YYYY`),
- `n` is the number of people (i.e., distinct email addresses) who received an email from `s` during month `m`,
- there is no other month (in the input dataset) where `s` sent strictly more than `n` emails (*to* `n` *distinct emails*).

The output should be sorted in descending order on `n` first, then `s`.

Note that given a `datetime` object you can extract its month (or year) using its `month` (or `year`) attribute, e.g., `myObj.month` (or `myObj.year`).

Task 3: 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 task, you will convert the Email network extracted in Task 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 Task 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`.

Task 4: 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.

Task 4.1 (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 Task 2.

Task 4.2 (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 Task 3.