ORIE 5270: BIG DATA TECHNOLOGIES
HOMEWORK 5 – DUE: FRIDAY 05/14/2021

Instructions. Submit on Gradescope by **Fri., May 14th** at **11:59pm (midnight) US EDT**).

Collaboration between students is allowed as long as it is limited to high-level details and/or debugging help. The work you submit must be your own work. Make sure to acknowledge any collaborations and/or sources that helped you obtain your solutions.

Reminder: if you are unable to attend any of the regular office hours, please contact us on Campuswire to arrange for a short 1-1 session on Zoom.

Problem 1 (MapReduce for π (30%))**.** In this problem, you will use Monte Carlo simulation to estimate the value of the mathematical constant π. When we wish to estimate an unknown quantity that is the expectation of some real-valued process, we can do so via Monte-Carlo simulation by generating N independent trials of that process and averaging their output. The larger N is, the more accurate our estimate will be.

For estimating π, the algorithm works as follows:

1. Generate N random vectors $z^i \in \mathbb{R}^2$ which are sampled from the uniform distribution in the unit square, $[-1,1] \times [-1,1]$.
2. For each of those random numbers, output `1` if $\sqrt{(z_1^i)^2 + (z_2^i)^2} \leq 1$, and `0` otherwise.
3. Let N_1 be the number of 1's from the previous step; our estimate of π is $\hat{\pi} = \frac{4N_1}{N}$.

Why does this algorithm work? The second step of the algorithm outputs 1 only if the sample z^i falls in the unit circle. In other words, the probability that we increment our count N_1 is equal to the probability that z^i falls inside the unit circle. The unit circle has volume $\pi r^2 = \pi$, while the unit square has volume $2 \times 2 = 4$. Therefore, the probability that we output 1 is equal to $\frac{\pi}{4}$, so $\mathbb{E}[N_1] = \frac{N\pi}{4}$, which is why we scale the estimate by 4.

Write a MapReduce script called `mrpi.py` to estimates π via Monte-Carlo simulation, with the help of the `mrjob` library. In particular, one should be able to run your script like below:

```
$ python mrpi.py samples.txt
```

Here, `samples.txt` is a file with one integer per line that dictates how many samples each `mapper` function should simulate. For example:

```
$ cat samples.txt
```

```
15
20
15
25
...
```

With the above file, the first instance of the mapper function should generate 15 samples, the second instance should generate 20 samples, and so on.

Problem 2 (MapReduce for co-author citations (30%))**.** Suppose you are given an input file called `authors.txt` where each line contains 3 fields:

1. The first field is the full name of a single author;
2. The second field is the name of a paper they have authored;
3. The third field is the number of citations of the paper.

The 3 fields are separated by commas; an example of such a file follows below:

```
$ cat authors.txt
Author 1, Paper A, 57
Author 1, Paper C, 19
Author 2, Paper B, 32
Author 2, Paper C, 19
Author 3, Paper A, 57
Author 3, Paper C, 19
Author 4, Paper B, 32
Author 4, Paper D, 29
```

Download the file `mrcoauthors.py` and complete it by writing a MapReduce program that outputs a sequence of `(key, value)` pairs as follows: the `key` field should be a pair of co-authors and the `value` field should contain the *total number of citations* over the papers they have co-authored. For example, given the above input, your program should behave as follows:

```
$ python mrcoauthors.py authors.txt
(Author 1, Author 2), 19
(Author 1, Author 3), 76
(Author 2, Author 3), 19
(Author 2, Author 4), 32
```

Here the first pair of authors have only co-authored Paper C with 19 citations, but Author 1 and Author 3 have co-authored papers A and C with $19 + 57 = 76$ citations total. **Notice**

that the pair `(Author 3, Author 4)` does not appear at all as a key since the two authors have not co-authored any papers to gether.

Hint: design your first mapper function so that you group authors and/or citations by paper title.

Problem 3 (A simple SQL database (40%))**.** In this problem, you will design a simple SQL database to help Cornell ORIE keep track of course enrollment and TA assignments.

The database should keep track of `Courses`, `Faculty` and `Students`. For each of these, we need the following information:

- `Courses` should have a course number, course name, subject (one of "Optimization", "Probability" and "Statistics"), and semester they are taught in (Fall vs. Spring). In addition, every course is taught by *one* faculty member, supported by any number of student TAs, and has any number of students enrolled.
- `Faculty` should have an associated first name, last name, netID, and specialization (one of "Optimization", "Probability" and "Statistics"). In addition, a faculty member may be teaching zero, one or more courses.
- `Students` have an associated first name, last name, netID and semester they are currently enrolled in. In addition:
  1. every student can be enrolled in one or more courses; and
  2. every student can serve as a TA for one or more courses.

Part 1: Design a database schema that captures the above information. In particular:

- Create all the tables necessary for holding course, student and faculty information;
- Identify any relations between different entities, as well as whether they are 1:1, 1:N or M:N.
- Determine which columns, if any, should be primary / foreign keys, unique or not, allowed to be `NULL`, etc.;
- For all M:N relations, create the necessary junction tables to represent them.

Download the file `mk_tables.sql` and add all necessary SQL commands for creating the above database schema. The file `mk_tables.sql` already contains most definitions of the tables, but is missing primary / foreign key information, junction tables, etc. To create a database called `courses.db` with this schema, you can run

```
$ sqlite courses.db < mk_tables.sql
```

Note: depending on your operating system, your SQLite executable might be called `sqlite3` instead of `sqlite`.

Part 2: Create an SQL script `populate.sql` that does the following:

- Inserts at least 3 courses into the `Courses` table;
- Inserts at least 3 faculty into the `Faculty` table;
- Inserts at least 10 students into the `Students` table;
- Assigns 2 students as TAs to the first course you created, and 1 student as TA to each of the other courses. **At least one** of the student TAs should be a TA for multiple courses.
- Assigns a faculty to teach each course.
- Enrolls students in various courses (the individual choices are up to you).

To populate the previously created `courses.db` database using the above script, you can run

```
$ sqlite courses.db < populate.sql
```

Part 3: Create an SQL script `queries.sql` that does the following:

- selects the last name and netID of all faculty whose specialization is "Probability" or "Statistics";
- selects the netIDs of all students that are TAs for any class whose subject is "Optimization";
- selects the netIDs of all students that are TAs for a course that they are also enrolled in at the same time;
- selects all courses with course number bigger than 4000 as well as the last name of the faculty teaching each of these courses.