CSC 343, Exam III Review

Exam II Notes

- You may bring one page of notes (front and back) to the exam. This page may be handwritten or typed.
- Computer access will not be permitted during the exam.
- Cell phones must be put away at all times.
- Don't hesitate to contact me if you have any questions!

Exam II Outline

- Python programming concepts
  - Named and anonymous functions (lambda notation)
  - map, filter, and reduce
  - dictionaries and JSON-formatted strings
- Spark Concepts
  - RDDs and lineage
  - Pair RDDs
  - Partitions
  - Persistence
- PySpark
  - Creating RDDs using *parallelize*, *textFile*, and *wholeTextFiles*
  - Transformations, including *map*, *flatMap*, *filter, reduce,* and *distinct*
  - Pair RDDs and transformations including *keyBy*, *groupByKey*, and *join*
  - Actions including *count*, *collect*, *take*, *first*, and *countByKey*
  - Additional functions, including *persist* and *numPartitions*

Exam II Practice

1. Write a function that takes someone's first and last name and returns a string of the form 'last, first'
2. The object *courses* below contains a list of JSON-formatted strings. Use PySpark to create an RDD where each record is a JSON-formatted string, then do the following:

```
courses = ["""{"Subj":"CSC", "Crse": "450"}""",
           """{"Subj":"CSC", "Crse": "315"}""",
           """{"Subj":"MAT", "Crse": "216"}"""]
```

   a. Transform each record into a python dictionary
   b. Filter out the records where the subject is not CSC
   c. Transform the remaining records into strings with the format CSC-450.

d. Starting from (a), transform the RDD into a pair RDD where the key is the subject and the value is the course. For each subject, count the number of courses and display the subject and corresponding frequency.

The file *report.txt* is a plain text version of the Mueller Report (https://www.justice.gov/storage/report.pdf), released to the public on April 18, 2019, which reports on the "Investigation into Russian Interference in the 2016 Presidential Election". Note: because the format of the original report is a scanned PDF, mistakes have been introduced when converting from scanned PDF to plain text file. The exercises below will give you insight into the report, but the specific results may not be correct.

1. Create an RDD from *report.txt* where each line is a record in the RDD.
2. How many records are in the RDD (e.g., how many lines are in the file)?
3. Output the first 10 lines.
4. Transform this RDD so that each record contains only a single word.
5. What is the word count of the file (the total number of records)? How many unique words (records) are there?
6. Transform the RDD from (4) to keep only words that are more than 4 letters long.
7. Now apply the appropriate transformations to find the 20 most frequent words and their corresponding frequencies.
8. The most common type of redaction was for "Harm to Ongoing Matter", or "HOM". Use Spark's map and reduce functions to find the total number of redactions that contain "Harm to Ongoing" or "HOM". Note: python strings have a *count* method that will count the number of occurrences of a set of characters in a string.

Spark Concepts:

9. What is the output when the code below is run, and why, assuming the file is deleted where indicated? Assume that *file.txt* has 10 lines and 34 words in it.

```
# read in the text file
rdd = sc.textFile('file.txt')

# count the number of lines
print(rdd.count())

# transform to RDD where each record is a word
rddWords = rdd.flatMap(lambda x: x.split())

# suppose that the file 'file.txt' is now deleted

# count the number of words
print(rddWords.count())
```

10. Create a diagram similar to the one on page 22 of the Spark Parallel Processing notes, to label the tasks and stages for the following spark operation. Also show the records that are created in each RDD. Assume that the data from the file is stored in 2 partitions, and that the file contains the text:

Jeff	Fr	18
Ali	Jr	22
Steve	Jr	21
Mary	Fr	17

```
sc.textFile('ages.txt', numPartitions = 2) \
.map(lambda x: x.split('\t')) \
.map(lambda x: (x[1], int(x[2]))) \
.groupByKey(numPartitions = 1) \
.mapValues(lambda x: sum(x)/len(x)) \
.collect()
```

For example, the first RDD has two partitions and is given by:

RDD1

'Jeff\tFr\t18' 'Ali\tJr\t22'
'Steve\tJr\t21' 'Mary\tFr\t17'