Assignment 2: Algorithms in MapReduce

(Adapted from Bill Howe's programming assignment, Coursera course Introduction to Data Science)

In this assignment, you will be designing and implementing MapReduce algorithms for a variety of common data processing tasks.

The goal of this assignment is to give you experience "thinking in MapReduce." We will be using small datasets that you can inspect directly to determine the correctness of your results and to internalize how MapReduce works.

Python MapReduce Framework

You will be provided with a python library called MapReduce.py that implements the MapReduce programming model. The framework faithfully implements the MapReduce programming model, but it executes entirely on a single machine -- it does not involve parallel computation.

Here is the word count example discussed in class implemented as a MapReduce program using the framework:

```
# Part 1
mr = MapReduce.MapReduce()

# Part 2
def mapper(record):
    # key: document identifier
    # value: document contents
    key = record[0]
    value = record[1]
    words = value.split()
    for w in words:
      mr.emit_intermediate(w, 1)

# Part 3
def reducer(key, list_of_values):
    # key: word
    # value: list of occurrence counts
    total = 0
    for v in list_of_values:
      total += v
    mr.emit((key, total))

# Part 4
inputdata = open(sys.argv[1])
mr.execute(inputdata, mapper, reducer)
```

In Part 1, we create a MapReduce object that is used to pass data between the map function and the reduce function; you won't need to use this object directly.

In Part 2, the mapper function tokenizes each document and emits a key-value pair. The key is a word formatted as a string and the value is the integer 1 to indicate an occurrence of word.

In Part 3, the reducer function sums up the list of occurrence counts and emits a count for word. Since the mapper function emits the integer 1 for each word, each element in the list_of_values is the integer 1.

The list of occurrence counts is summed and a (word, total) tuple is emitted where word is a string and total is an integer.

In Part 4, the code loads the json file and executes the MapReduce query which prints the result to stdout.

Submission Details

For each problem, you will turn in a python script, similar to wordcount.py, that solves the problem using the supplied MapReduce framework.

When testing, make sure MapReduce.py is in the same directory as the solution script.

To allow you to test your programs, sample data will be provided in the data folder, and the corresponding solutions will be provided for each problem in the solutions folder.

Your python submission scripts are required to have a mapper function that accepts at least 1 argument and a reducer function that accepts at least 2 arguments. Your submission is also required to have a global variable named mr which points to a MapReduce object.

If you solve the problems by simply replacing the mapper and reducer functions in wordcount.py, then this condition will be satisfied automatically.

What you will turn in:

- These five files (Note: you **don't** need to use a .zip file):
  - Program that implements a MapReduce algorithm to create inverted index called: `lastname_firstname_inverted_index.py`
  - Program that implements a MapReduce algorithm to count the number of friends for each person called: `lastname_firstname_friend_count.py`
  - Program that implements a MapReduce algorithm to identify asymmetric friendships called: `lastname_firstname_asymmetric_friendships.py`
  - Program that implements a MapReduce algorithm that removes the last 10 characters from each string of nucleotides, removing any duplicates generated, called: `lastname_firstname_unique_trims.py`
  - Program that implements a MapReduce algorithm for matrix multiplication called: `lastname_firstname_multiply.py`

Problem 1

Create a python program called `lastname_firstname_inverted_index.py` that implements a mapReduce algorithm to generate an Inverted index. Given a set of documents, an inverted index is a dictionary where each word is associated with a list of the document identifiers in which that word appears. An inverted index is used in search engines to match a search term with pointers to all the documents that contain that search term.

Mapper Input

The input is a 2 element list: `[document_id, text]`, where `document_id` is a string representing a document identifier and `text` is a string representing the text of the document. The document text may have words in upper or lower case and may contain punctuation. You should treat each token as if it was a valid word; that is, you can just use value.split() to tokenize the string.

Reducer Output

The output should be a (word, document ID list) tuple where word is a String and document ID list is a list of Strings.

You can test your solution to this problem using the data file books.json:

```
python lastname_firstname_inverted_index.py books.json
```

You can verify your solution against inverted_index.json.

Problem 2

Consider a simple social network dataset consisting of a set of key-value pairs `(person, friend)` representing a friend relationship between two people. Create a python program called `lastname_firstname_friend_count.py` that implements a MapReduce algorithm to count the number of friends for each person.

Map Input

Each input record is a 2 element list `[personA, personB]` where personA is a string representing the name of a person and personB is a string representing the name of one of personA's friends. Note that it may or may not be the case that the personA is a friend of personB.

Reduce Output

The output should be a pair `(person, friend_count)` where person is a string and friend_count is an integer indicating the number of friends associated with person.

You can test your solution to this problem using the data file friends.json:

```
$ python lastname_firstname_friend_count.py friends.json
```

You can verify your solution by comparing your result with the file friend_count.json.

Problem 3

The relationship "friend" is often symmetric, meaning that if I am your friend, you are my friend.

Create a python program called `lastname_firstname_asymmetric_friendships.py` that implements a MapReduce algorithm to identify asymmetric friendships. In other words, if `[personA, personB]` appears in the list of friends, check whether `[personB, personA]` is also on the list.

If **only one** of the friendships appears (an asymmetric relationship), emit the pair of names in both orders. If the relationship is symmetric, then produce no output.

Map Input

Each input record is a 2 element list `[personA, personB]` where personA is a string representing the name of a person and personB is a string representing the name of one of personA's friends. Note that it may or may not be the case that the personA is a friend of personB.

Hint: Think about emitting a unique key associated with a pair of names.

Reduce Output

The output should be the full symmetric relation for only those input pairs that were originally asymmetric. When you identify an asymmetric pair, you will emit BOTH (person, friend) AND (friend, person). Be careful to avoid producing duplicates. For symmetric pairs, produce no output.

You can test your solution to this problem using the data file friends.json:

```
$ python lastname_firstname_asymmetric_friendships.py friends.json
```

You can verify your solution by comparing your result with the file asymmetric_friendships.json.

Problem 4

Consider a set of key-value pairs where each key is sequence id and each value is a string of nucleotides, e.g., GCTTCCGAAATGCTCGAA....

Create a python program called `lastname_firstname_unique_trims.py` that uses a MapReduce algorithm that removes the last 10 characters from each string of nucleotides, removing any duplicates generated.

Map Input

Each input record is a 2 element list `[sequence id, nucleotides]` where sequence id is a string representing a unique identifier for the sequence and nucleotides is a string representing a sequence of nucleotides

Reduce Output

The output from the reduce function should be the unique trimmed nucleotide strings.

You can test your solution to this problem using the data file dna.json:

```
$ python lastname_firstname_unique_trims.py dna.json
```

You can verify your solution by comparing your result with the file unique_trims.json.

Problem 5

Assume you have two matrices A and B in a sparse matrix format, where each record is of the form i, j, value. Create a python program called `lastname_firstname_multiply.py` that implements a MapReduce algorithm to compute the matrix multiplication A x B.

For this problem, you can assume that the matrices to be multiplied are 5x5 matrices with integer values.

Hint: Each Map step may need to emit multiple outputs with different keys to guarantee that the right information gets to each Reduce step.

Map Input

The input to the map function will be a row of a matrix represented as a list. Each list will be of the form `[matrix, i, j, value]` where matrix is a string and i, j, and value are integers.

The first item, matrix, is a string that identifies which matrix the record originates from. This field has two possible values: "a" indicates that the record is from matrix A and "b" indicates that the record is from matrix B

Reduce Output

The output from the reduce function will also be a row of the result matrix represented as a tuple. Each tuple will be of the form (i, j, value) where each element is an integer.

You can test your solution to this problem using the data file matrix.json:

```
$ python lastname_firstname_multiply.py matrix.json
```

You can verify your solution by comparing your result with the file multiply.json.