Algorithms in MapReduce: Instructions

When you're ready to submit your solution, go to the assignments list.

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

The MapReduce programming model (and a corresponding system) was proposed in a 2004 paper from a team at Google as a simpler abstraction for processing very large datasets in parallel. 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. In the next assignment, you will have the opportunity to use a MapReduce-based system to process the very large datasets for which is was designed.

As always, the first thing to do is to update your provided course materials using git pull. These resources may have changed since the last time you interacted with the datasci_course_materials repository. Github Instructions.

Next, review the lectures to make sure you understand the programming model.

You may also want to experiment running your queries on the JSMapReduce service for this assignment to see what it would be like to run a MapReduce query on a cluster of machines.

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 folder as the solution script.

Solution data 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 is guaranteed.

Problem 1

Create 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.

Mapper Input

The input is a 2 element list: [document_id, text]

document_id: document identifier formatted as a string

text: text of the document formatted as a string

The document text may have words in various cases or elements of punctuation. Do not modify the string, and treat each token as if it was a valid word. (That is, just use value.split())

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 books.json:

    python inverted_index.py books.json

You can verify your solution against inverted_index.json.

Problem 2

Implement a relational join as a MapReduce query

Consider the query:

```
SELECT *

FROM Orders, LineItem

WHERE Order. ORDER_ID  = Line Item. ORDER_ID
```

Your MapReduce query should produce the same information as this SQL query. You can consider the two input tables, Order and LineItem, as one big concatenated bag of records which gets fed into the map function record by record.

Map Input

The input will be database records formatted as lists of Strings.

Every list element corresponds to a different field in it's corresponding record.

The first item(index 0) in each record is a string that identifies which table the record originates from. This field has two possible values:

    'line_item' indicates that the record is a line item.

'order' indicates that the record is an order.

The second element(index 1) in each record is the order_id.

LineItem records have 17 elements including the identifier string.

Order records have 10 elements including the identifier string.

Reduce Output

The output should be a joined record.

The result should be a single list of length 27 that contains the fields from the order record followed by the fields from the line item record. Each list element should be a string.

You can test your solution to this problem using records.json:

```
python join.py records.json
```

You can verify your solution against join.json.

Problem 3

Consider a simple social network dataset consisting of key-value pairs where each key is a person and each value is a friend of that person. Describe a MapReduce algorithm to count he number of friends each person has.

Map Input

The input is a 2 element list: [personA, personB]

personA: Name of a person formatted as a string

personB: Name of one of personA's friends formatted as a string

This implies that personB is a friend of personA, but it does not imply that personA is a friend of personB.

Reduce Output

The output should be a (person, friend count) tuple.

person is a string and friend count is an integer describing the number of friends 'person' has.

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

```
python friend_count.py friends.json
```

You can verify your solution against friend_count.json.

Problem 4

The relationship "friend" is often symmetric, meaning that if I am your friend, you are my friend. Implement a MapReduce algorithm to check whether this property holds. Generate a list of all non-symmetric friend relationships.

Map Input

The input is a 2 element list: [personA, personB]

personA: Name of a person formatted as a string

personB: Name of one of personA's friends formatted as a string

This implies that personB is a friend of personA, but it does not imply that personA is a friend of personB.

Reduce Output

The output should be a list of (person, friend) and (friend, person) tuples for each asymmetric friendship.

Only one of the (person, friend) or (friend, person) output tuples will exist in the input. This indicates friendship asymmetry.

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

```
python asymmetric_friendships.py friends.json
```

You can verify your solution against asymmetric_friendships.json.

Problem 5

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

Write a MapReduce query to remove the last 10 characters from each string of nucleotides, then remove any duplicates generated.

Map Input

The input is a 2 element list: [sequence id, nucleotides]

sequence id: Unique identifier formatted as a string

nucleotides: Sequence of nucleotides formatted as a string

Reduce Output

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

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

python unique_trims.py dna.json

You can verify your solution against unique_trims.json.

Problem 6

Assume you have two matrices A and B in a sparse matrix format, where each record is of the form i, j, value. Design a MapReduce algorithm to compute matrix multiplication: A x B

Map Input

The input to the map function will be matrix row records formatted as lists. Each list will have the format [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

‘b’ indicates that the record is from matrix B

Reduce Output

The output from the reduce function will also be matrix row records formatted as tuples. Each tuple will have the format (i, j, value) where each element is an integer.

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

python multiply.py matrix.json

You can verify your solution against multiply.json.