Aims

This exercise aims to get you to apply the order inversion and value-to-key conversion design patterns for MapReduce programming you have learned in Chapter 3.1. If you have not finished solving the problems in Lab 3, please keep working on them, and then move to Lab 4.

Create a folder "Lab4" and put all your codes written in this week's lab in this folder and keep a copy for yourself after you have finished all the problems.

For all problems, using the following codes to tokenize a line of document:

```
import re

words = re.split("[ *$&#/\t\n\f\"\'\\,.:;?!\[\](){}<>~\-_]", line.lower())
```

Examples on Partitioner and Comparator

Hadoop Partitioner Class

Hadoop has a library class, KeyFieldBasedPartitioner, that is useful for many applications. This class allows the MapReduce framework to partition the map outputs based on certain key fields, not the whole keys. For example:

```
mapred streaming \
  -D stream.map.output.field.separator=. \
  -D stream.num.map.output.key.fields=4 \
  -D mapreduce.map.output.key.field.separator=. \
  -D mapreduce.partition.keypartitioner.options=-k1,2 \
  -D mapreduce.job.reduces=12 \
  -input myInputDirs \
  -output myOutputDir \
  -mapper /bin/cat \
  -reducer /bin/cat \
  -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

Here, -D stream.map.output.field.separator=. tells Hadoop that "." is the separator for keys and values, and -D stream.num.map.output.key.fields=4 means that the prefix before the fourth "." is used as the key, and the remaining part is used as the value.

The map output keys of the above MapReduce job normally have four fields separated by ".". However, the MapReduce framework will partition the map outputs by the first two fields of the keys using the -D mapred.text.key.partitioner.options=-k1,2 option. Here, -D mapreduce.map.output.key.field.separator=. specifies the separator for the partition. This guarantees that all the key/value pairs with the same first two fields in the keys will be partitioned into the same reducer.

This is effectively equivalent to specifying the first two fields as the primary key and the next two fields as the secondary. The primary key is used for partitioning, and the combination of the primary and secondary keys is used for sorting. A simple illustration is shown here:

Output of map (the keys)

```
11.12.1.2
11.14.2.3
11.11.4.1
11.12.1.1
11.14.2.2
```

Partition into 3 reducers (the first 2 fields are used as keys for partition)

```
11.11.4.1
-----------
11.12.1.2
11.12.1.1
-----------
11.14.2.3
11.14.2.2
```

Sorting within each partition for the reducer (all 4 fields used for sorting)

```
11.11.4.1
-----------
11.12.1.1
11.12.1.2
-----------
11.14.2.2
11.14.2.3
```

Hadoop Comparator Class

Hadoop has a library class, KeyFieldBasedComparator, that is useful for many applications. This class provides a subset of features provided by the Unix/GNU Sort. For example:

```
mapred streaming \
  -D
mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapreduce.lib.parti
tion.KeyFieldBasedComparator \
  -D stream.map.output.field.separator=. \
  -D stream.num.map.output.key.fields=4 \
  -D mapreduce.map.output.key.field.separator=. \
  -D mapreduce.partition.keycomparator.options=-k2,2nr \
  -D mapreduce.job.reduces=1 \
  -input myInputDirs \
  -output myOutputDir \
  -mapper /bin/cat \
  -reducer /bin/cat
```

The map output keys of the above Map/Reduce job normally have four fields separated by “.”. However, the Map/Reduce framework will sort the outputs by the second field of the keys using the -D mapreduce.partition.keycomparator.options=-k2,2nr option. Here, -n specifies that the sorting is numerical sorting and -r specifies that the result should be reversed. A simple illustration is shown below:

Output of map (the keys)

```
11.12.1.2
11.14.2.3
11.11.4.1
11.12.1.1
11.14.2.2
```

Sorting output for the reducer (where the second field is used for sorting)

```
11.14.2.3
11.14.2.2
11.12.1.2
11.12.1.1
11.11.4.1
```

Partitioner and comparator in mrjob

Similarly, you can use the same configurations in mrjob. Given the input as in above examples (https://webcms3.cse.unsw.edu.au/COMP9313/23T3/resources/92046), if you want to partition the mapper output keys according to the first two fields (separated by ".") rather than the entire key (four fields in total), you can configure your mrjob as follows:

```
from mrjob.job import MRJob

class TestPartitioner(MRJob):

    def mapper(self, _, line):
        yield line,''

    def reducer(self, key, values):
        yield key, ';'.join(values)

    SORT_VALUES = True

    JOBCONF = {
      'mapreduce.map.output.key.field.separator':'.',
      'mapreduce.job.reduces':2,
      'mapreduce.partition.keypartitioner.options':'-k1,2'
    }
if __name__ == '__main__':
    TestPartitioner.run()
```

Please note that this should run on Hadoop (with "-r hadoop" in the command, together with "-o hdfs://localhost:9000/user/comp9313/output" to store the results to HDFS), and the line "SORT_VALUES = True" is necessary. You will see that the 5 lines are sent to two reducers for processing, partitioned by the first two fields.

Besides controlling the partitioning, you can also tell Hadoop how to sort the mapper output using the comparator class. Still use the same input (each line has 4 fields separated by "," and the entire line is used as the key). Assume you want to partition based on the first two fields, and the sort order is first based on the first two fields and then based on the fourth field in descending order by numerical values, you can use the mrjob codes as follows:

```
from mrjob.job import MRJob

class PartitionerAndComparator(MRJob):

    def mapper(self, _, line):
        yield line,''

    def reducer(self, key, values):
        yield key, ';'.join(values)

    SORT_VALUES = True

    JOBCONF = {
      'mapreduce.map.output.key.field.separator':'.',
      'mapreduce.job.reduces':2,
      'mapreduce.partition.keypartitioner.options':'-k1,2',

'mapreduce.job.output.key.comparator.class':'org.apache.hadoop.mapreduce.lib.pa
rtition.KeyFieldBasedComparator',
      'mapreduce.partition.keycomparator.options':'-k1,2 -k4,4nr'
    }
if __name__ == '__main__':
    PartitionerAndComparator.run()
```

You still need to run on Hadoop. 'mapreduce.partition.keypartitioner.options':'-k1,2' specifies the partition strategy, and 'mapreduce.partition.keycomparator.options':'-k1,2 -k4,4nr' specifies the sort order. You can compare the results with those obtained by running TestPartitioner.

Problem 1. Computer Relative Frequency Using Order Inversion

The problem is to compute the relative frequency $f(w_j|w_i)$, i.e.,

$$f(w_j|w_i) = \frac{N(w_i, w_j)}{\sum_{w'} N(w_i, w')}$$

Here, N(., .) indicates the number of times a particular co-occurring term pair is observed in the corpus. We need the count of the term co-occurrence, divided by the marginal (the sum of the counts of the conditioning variable co-occurring with anything else).

In this problem, we consider the **nonsymmetric** co-occurrence. That is, w_i and w_j co-occur if w_j appears after w_i in the same line, which is defined the same as in Problem 2 of Lab 3.

Hadoop Streaming: Create new python scripts NSRelativeFreq_mapper.py and NSRelativeFreq_reducer.py in the folder Lab4.

mrjob: Create a new script “mr_NSRelativeFreq.py” in the folder Lab4 and solve this problem.

Your output should be in format of (w_i, w_j, $f(w_j|w_i)$).

Hints:

1. Refer to slides 16-20 of Chapter 3.1.
2. The mapper only needs to be modified a little bit. Just emit one more special key for a pair of terms. The problem is, what is the output key of the mapper? How to guarantee that the order is correct, and thus the special key can be sent to the reducer first (You can still use Text to wrap a pair of terms)?
3. Can you use the reducer as the combiner in this problem? How to write a combiner to aggregate partial results?
4. How to specify the configuration in JOBCONF?

You can use the results of Problem 2 in Lab 3 to verify the correctness of your output (e.g., check the pairs where the first term is "zounds").

Question: What will happen if you set the number of reducers more than 1?

Try to work on the following problem: to see the effects of the partitioner, we want to set the number of reducers to 2 using the option `mapreduce.job.reduces=2` in JOBCONF. Just consider how to guarantee all key-value pairs relevant to the first term are sent to the same reducer.

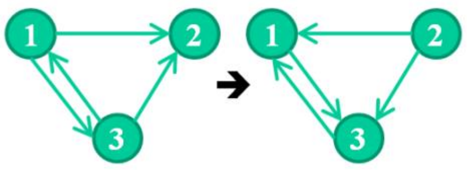
Hint:

1. Please refer to the examples provided at the beginning of this lab to find out how to use the library partitioner class embedded in Hadoop.

Problem 2. Reverse Graph Edge Directions

The problem is to reverse graph edge directions of the input directed graph and sort the output adjacency list in ascending node order. You can refer to Slide 63 of Chapter 3.1.

mrjob: Create a script "mr_ReverseDirect.py" in the folder Lab4 and then solve this problem using the value-to-key design pattern.



Input files:

In the input file, each line is in the format of: "FromNodeId**\t**ToNodeId$_1$ ToNodeId$_2$ ToNodeId$_3$ … ToNodeId$_n$". In the above example, the input file is:

```
3\t1 2
1\t2 3
```

Output files:

The format of the output file is the same as that of the input file. Moreover, the lines are required to be sorted ascendingly according to their FromNodeIds. In each line, the ToNodeIds are also required to be sorted in accending order.

```
1\t3
2\t1 3
3\t1
```

Hints:

1. Because of the requirement of ascending order of ToNodeIds, you are required to use secondary sort to solve this problem. Please refer to the provided examples at the beginning of this lab to find out how to use the partitioner and comparator in MRJob by configuring JOBCONF.

Problem 3. Find the Top-k Frequent Terms

The problem is to find the top-k frequent terms in a given textual dataset. Still use pg100.txt as the input file. If two terms have the same frequency, sort them alphabetically.

mrjob: Create a script "mr_TopkTerms.py" in the folder Lab4 and then solve this problem using multiple MRSteps.

Hints:

1. Refer to slide 64 of Chapter 3.1
2. k is a parameter received from the command. You can access the value in your program like "k = jobconf_from_env('myjob.settings.topk')" (use "from mrjob.compat import jobconf_from_env" in your code). Then, you can pass the value of k by running: `python3mr_TopkTerms.py.py -r hadoop hdfs_input -o hdfs_output --jobconf myjob.settings.topk=20`
3. You need to use two MRSteps to solve this problem, and each step can have one jobconf.
4. You can use a heap ("import heapq") to find out the top-k efficiently.

Solutions of the Problems

I hope that you can finish all problems by yourself, since the hints are already given. All the source codes will be published in the course homepage on Friday in the same week.