CSE427S

HOMEWORK 5

M. Neumann

Due: THU 4 OCT 2018 4PM

Getting Started

Update your SVN repository.

When needed, you will find additional materials for *homework x* in the folder `hwx`. So, for the current assignment the folder is `hw5`.

Hint: You can **check your submission** to the SVN repository by viewing `https://svn.seas.wustl.edu/repositories/<yourwustlkey>/cse427s_fl18` in a web browser.

SUBMISSION INSTRUCTIONS

WRITTEN:

- all written work needs to be submitted electronically in *pdf format*[1] via GRADESCOPE
- provide the following information on *every* page of your `pdf` file:
  - name
  - student ID
  - wustlkey (your `wustlkey` indicates the location (SVN repository) for your code submissions; without this information we will not be able to grade your code!)
- start every problem on a *new page*
- **FOR GROUPS**: make a **group submission** on GRADESCOPE and provide the following information for **all group members** on every page of your `pdf` file: names, student IDs, and location of code submissions (**one** student's `wustlkey`)[2]

CODE:

- code needs to be submitted via SVN repository commit (detailed submission instructions are provided whenever code submissions are required)
- make sure to always use the required *file name(s)* and *submission format(s)*
- **comment your code** to receive maximum credit

[1] Please, **type your solutions** or use **clear hand-writing**. If we cannot read your answer, we cannot give you credit nor will we be able to meet any regrade requests concerning your writing.

[2] It is sufficient to commit code to one SVN repository. If you do not specify the repository for your group's code submission clearly, we will only check the first repository, sorting wustlkeys alphabetically.

Problem 1: Custom WritableComparable (30%)

In this problem you will write a simple program that counts the number of occurrences of each full name (i.e., first and last name) in a list of names.

The stub are provided in:

```
~/workspace/word_co_writable/src/stubs
```

Use the following **test input** to test your implementation:

```
Smith Joe 1963-08-12 Poughkeepsie, NY
Murphy Alice 2004-06-02 Berlin, MA
Smith Joe 1832-01-20 Sacramento, CA
```

The output for this test input should look like this:

```
(Murphy,Alice) 1
(Smith,Joe) 2
```

(a) Implement a custom `WritableComparable` type holding two strings. Add your own comments to the code to receive maximum credit.

Note that Eclipse automatically generated the hashCode and equals methods in the stub file. You can generate these two methods in Eclipse by right-clicking in the source code and choosing "Source" -> "Generate hashCode() and equals()".

(b) Implement a MAPREDUCE program counting the number of occurrences of each (full) name (i.e., first and last name) in a list of names provided as given in the test input above. Add your own comments to the code to receive maximum credit.

Your MAPREDUCE program requires a Reducer that sums the number of occurrences of each key. This is very much the same as the SumReducer previously used in word-count. So, reuse the SumReducer with minor adaptations.

(c) Test your code using local job runner on the following input:

```
~/training_materials/developer/data/nameyeartestdata
```

Provide the last 7 lines of the results file in your written submission.

Submit the implementation files of your Mapper, Reducer, Driver and Writable.
To add the .java files to your SVN repo before committing run:

```
$ svn add StringPairWritable.java
$ svn add WordCoMapper.java
$ svn add SumReducer.java
$ svn add WordCo.java
$ svn commit -m 'hw5 problem 1 submission' .
```

Problem 2: Secondary Sort (30%)

In this problem we will explore and modify the Secondary Sort problem from Lab 4. Recall that the goal of this secondary sort example is to sort the records first by last name in ascending order and then if the name is the same sort by birth year in descending order, so that the first entry per each last name is the *youngest* person of that last name.

(a) If our goal is to sort the values of a particular key, a **naive solution** would be to simply group by key and do the sorting in the reduce() function. Let's assume that the list of values *fits in memory* of the compute node running the Reducer.

- Let's also assume your MAPREDUCE job finishes successfully. What is a disadvantage of this approach? Consider parallel execution and partial sorting,
- Now, let's consider the case where your MAPREDUCE job breaks. Why does your job fail? HINT: Think about the amount of memory you need to sort the values.

(b) Run the Secondary Sort MAPREDUCE implementation form Lab 4 on the `nameyeartestdata` exactly as described in the lab instructions.

- Why is the output not globally sorted?
- What is the output of your MAPREDUCE program for the youngest person with last name Newton?

(c) Modify your MAPREDUCE program incorporate the following changes. The output should be *globally* sorted. Modify the `NameYearPartitioner` to achieve this. You can assume that the number of Reducers is fixed (same as described in the lab instructions). Do not use key distributions.

(d) You want to to output the *oldest* person instead of the youngest person. Do not change any comparators! You should be able to do this by merely modifying the job configuration. Describe the change you made to the job configuration.

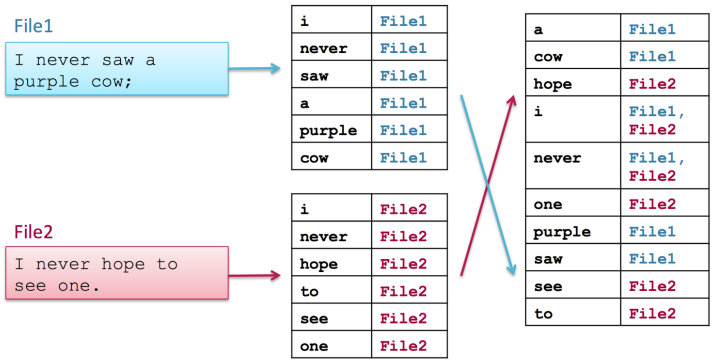
(e) Why do we need to implement the group comparator in the secondary sort example discussed in Lab 4?

To add the modified `NameYearPartitioner.java` file to your SVN repo before committing run:

```
$ svn add NameYearPartitioner.java
$ svn commit -m 'hw5 problem 2 submission' .
```

Problem 3: Creating an Inverted Index (40%)

An *index* is a list of contents (e.g., words) per file or document, i.e. it maps files or documents to its content. An *inverted index* maps contents (e.g., words) to files or documents. See also [MMDS]: Ch1.3.3 Indexes. Illustration:



Write a MAPREDUCE job that produces an inverted index. Use the stubs provided in:

```
~/workspace/inverted_index/src/stubs/
```

Use the input provided in the file `invertedIndexInput.tgz` located in

```
~/training_materials/developer/data/invertedIndexInput.tgz
```

Preparation: Extract the `invertedIndexInput` directory and upload it to HDFS .
When decompressed, this archive contains a directory of files; each is a Shakespeare play formatted as follows:

```
0 HAMLET
1
2
3 DRAMATIS PERSONAE
4
5
6 CLAUDIUS king of Denmark. (KING CLAUDIUS:)
7
8 HAMLET son to the late, and nephew to the present king.
...
```

Each line contains:

- *Line number*
- *separator*: a tab character
- *value*: the line of text

(a) Which `InputFormat` can be used to read this data directly as key value pairs? How can you retrieve the file name in a Mapper or Reducer? Provide the respective code in the `hw5.pdf` file.
HINT: Consult Table 8-7. *File split properties* in HTDG and note that the `getPath()` method has a `getName()` function.

(b) Implement an indexer that produces an index of **all** the words in the text. For each word, the index should have a list of all the locations (*filename @ line*) where the word appears. Transform all words to lowercase.

For the word *honeysuckle* the output should look like this:

```
honeysuckle 2kinghenryiv@1038,midsummernightsdream@2175
```

(c) What is the Mapper output for the following input lines from Hamlet:

```
282 have heaven and earth
133 there are more things in heaven and earth
```

Submit the implementation files of your Mapper and Reducer.
To add the .java files to your SVN repo before committing run:

```
$ svn add IndexMapper.java
$ svn add IndexReducer.java
$ svn commit -m 'hw5 problem 3 submission' .
```

Bonus Problem (5% up to a max. of 100%) - no group work!

Write a review for this homework and store it in the file `hw5_review.txt` provided in your SVN repository (and commit your changes). This file should only include the review, **no other information** such as name, wustlkey, etc. Remember that you are not graded for the content of your review, solely it's completion.

Provide the star rating for `hw5` via this link: `https://wustl.az1.qualtrics.com/jfe/form/SV_aaAc0ILmBleyKzj`.
You can only earn bonus points if you write a meaningful review of **at least 50 words** and provide the corresponding star rating. Bonus points are given to the **owner of the repository only** (no group work!).

Submit your review in the file **`hw5_review.txt`** provided in the `hw5` folder in your SVN repository. **To commit the file run:**

```
$ svn commit -m 'hw5 review submission' .
```

Take 1 minute to provide a star rating for this homework – your star rating should match