CSE427S

HOMEWORK 4

M. Neumann

Due: THU 26 SEPT 2019 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 `hw4`.

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
- start every problem on a *new page*
- **FOR GROUPS**: make a **group submission** on GRADESCOPE and provide names and student IDs for **all group members** on every page of your `pdf` file.

CODE:

- code needs to be submitted electronically in a *single .zip file* via GRADESCOPE (detailed submission instructions are provided under HOMEWORK 4 CODE below)
- 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.

Problem 1: Passing a Parameter via the Command Line (25%)

You will write an Average Word Length program that uses a Boolean parameter called *caseSensitive* to determine whether the Mapper class should treat upper and lower case letters as different (*case-sensitive*) or whether all letters should be converted to lower case (*case-insensitive*).
Preparation: Copy the package including the Driver, Mapper, and Reducer code you have written earlier from `~/workspace/avgwordlength/src/` to the following directory:

```
~/workspace/toolrunner/src/
```

Note on Copying Source Files
You can use Eclipse to copy a Java source file from one project or package to another by right-clicking on the file and selecting Copy, then right-clicking the new package and selecting Paste. If the packages have different names (e.g. if you copy from average-wordlength.solution to toolrunner.stubs), Eclipse will automatically change the package directive at the top of the file. If you copy the file using a file browser or the shell, you will have to do that manually.

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

```
No now is definitely not the best time
```

The case-sensitive output for this test input (using one Reducer) would be:

```
N    2.0
b    4.0
d    10.0
i    2.0
n    3.0
t    3.5
```

IMPLEMENTATION SPECIFICATIONS:

- Your `AvgWordLength` driver should use ToolRunner.
- Your `LetterMapper` should use the configuration parameter `caseSensitive` to either perform *case-sensitive* processing or *case-insensitive* processing by writing a `setup()` method to get the value of the parameter. Case sensitive processing should be your **default**.

(a) Write down the Mapper output, Reducer input, and Reducer output using the case-insensitive version of your program for the example **test input** provided above.

(b) Test your `AvgWordLength` driver on the **test input** provided above. Comment your code to receive maximum credit. Are there any differences in the job execution process(es)?

(c) Test your `LetterMapper` on the **test input**. Comment your code to receive maximum credit. Are there any differences in the job execution process(es)?

(d) Run your implementation (using the default parameter) on the `shakespeare` folder in HDFS (make sure the folder only contains the following files: poems, histories, comedies, and tragedies). What are the average word lengths for the following letters:

- A
- W
- a
- t
- z

(e) Run your implementation providing `caseSenstive=false` as a runtime parameter on the `shakespeare` folder in HDFS (make sure the folder only contains the following files: poems, histories, comedies, and tragedies). Provide the command for this the `hw4.pdf` file. Does the order of command line inputs matter (what happens if you specify the parameter after the output directory)? What are the average word lengths for the following letters:

- a
- w
- z

Problem 2: Word Count with a Partitioner (20%)

In this problem you will implement a Partitioner for the WordCount MAPREDUCE program that assigns positive, negative, and neutral words to different reducers.
The stubs are in the following directory:

```
~/workspace/partitioner/src/
```

In eclipse *right-click* on the stubs package and click **Refresh** to see the files.

IMPLEMENTATION SPECIFICATIONS:

- Implement a Partitioner using the stubs `SentimentPartitioner.java`. Your Partitioner should send each key-value pair to one out of three Reducers based on whether the key is appearing in the list of positive words (`positive-words.txt`), in the list of negative words (`negative-words.txt`), or in neither of them. The files including the word lists are in your SVN folder. Make sure you ignore all lines in the .txt files starting with `;`. Use **Distributed Cache** to access the files in the Partitioner. Remove the provided comments and add your own comments to your code to receive maximum credit.
- Observe the `SentimentPartitionerTest.java` program. You can use it to test if your partitioner works correctly. It contains an example to test for the correct assignment of one example positive word.

  Add the three tests specified in the stub file to the program to test your Partitioner and run it. Comment your code to receive maximum credit.

- Modify the WordCount Driver to implement ToolRunner (to be able to use DistributedCache) and to use your Partitioner and the appropriate number of Reducers. Modify the Mapper to transform the input words to all lower-case letters (i.e., make WordCount *case-insensitive*, **no** parameter handling via ToolRunner required).

(a) Run the MAPREDUCE job on the `shakespeare/poems` data (in HDFS). How many different *positive*, *negative*, and *neutral* words did Shakespeare use in his poems? Using the counts for positive and negative words compute the sentiment score s and the positivity score p using:

$$s = \frac{positive - negative}{postitive + negative}, \quad p = \frac{positive}{postitive + negative}.$$

Hint: use a pipe of `cat` in HDFS and `wc -l` to count the number of lines in the respective Reducer output files (you do not need to copy the result files from HDFS to your local file system).

Based on these statistics, do you think Shakespeare's poems suggest positive or negative emotions?

(b) Those sentiment statistics are not the best way to measure emotions from text. Describe two cases where it actually breaks (i.e., the above statistics are not a meaningful measure for the true emotion). Suggest an improved statistic or way to measure sentiments in text documents overcoming both limitations you identified.

continue to next page...

CSE427S

HOMEWORK 4 CODE

M. Neumann

Due: THU 26 SEPT 2019 4PM

SUBMISSION INSTRUCTIONS

CODE:

- create a **single zip file** named `hw4_YourName.zip` (or `hw4_YourName_YouPartnersName.zip` for groups) including all files listed below

  Example file names:

  `hw4_MarionNeumann.zip`
  or for teams:
  `hw4_MarionNeumann_AnnaLee.zip`
  or
  `hw4_AnnaLee_MarionNeumann.zip`

- submit the *zip file* to the `hw4` CODE assignment in GRADESCOPE
- your code will be **autograded** for *correctness* and **manually inspected** for *comments*

Problem 3: Passing a Parameter via the Command Line (15%)

(a) Submit your modified version of the `AvgWordLength` driver. Comment your code (i.e., add you *own* descriptive inline comments) to receive maximum credit.

(b) Submit your modified version of the `LetterMapper`. Comment your code (i.e., add you *own* descriptive inline comments) to receive maximum credit.

Files to be added to your `zip` file:

```
AvgWordLength.java
LetterMapper.java
```

Problem 4: Word Count with a Partitioner (40%)

(a) Submit your `SentimentPartitioner.java` program. Remove the provided comments and add your *own* comments to your code to receive maximum credit.

(b) Submit your `SentimentPartitionerTest.java` program. Comment your code (i.e., add you *own* descriptive inline comments) to receive maximum credit.

(c) Submit your modified WordCount Driver and Mapper Java programs. Comment your code (i.e., add you *own* descriptive inline comments documenting all changes) to receive maximum credit.

Files to be added to your `zip` file:

```
SentimentPartitioner.java
SentimentPartitionerTest.java
WordCount.java
WordMapper.java
```

Reflection (Bonus Problem for 5% up to a max. of 100%)

Reflect on your homework experience! Write a paragraph of at least 50 words to express your experiences and feelings when working on this assignment. Answer at least 2 of the following questions:

- What did you like/dislike about the assignment and why?
- What is the most important thing you learned and why do you think so?
- What surprised you, and why?
- Assuming you could start over again (with working on the assignment), what would you do differently and why?

Do not include/copy and past the questions into your reflection!

Submission Instructions

Store your reflection in the `hw4_reflection.txt` file provided in the `hw4`folder in your SVN repository and commit it.

This file should only include the reflection, **no other personal information** such as name, wustlkey, etc. reflections are not graded based on the content, but solely for completion.

To submit your reflection `cd` into the `hw4` folder and run:

```
$ svn commit -m 'hw4 reflection submission' .
```

Take 1 minute to provide an overall **star rating** for this homework.
Submit it via this link: `https://wustl.az1.qualtrics.com/jfe/form/SV_bIRKP1xobpO8yIB`.

Grading - no group work!

You can only earn bonus points if you write a *meaningful* **reflection** of **at least 50 words** answering at least 2 of the prompted questions and provide the corresponding **star rating**. You will **not** be graded on what your reflection says and the number of stars you assign, but rather solely the completion of it.
Bonus points are given to the **owner of the repository only**. No group work!.