CSE427S

HOMEWORK 6

M. Neumann

Due: THU 10 OCT 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 `hw6`.

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 6 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: Custom WritableComparable (10%)

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
```

IMPLEMENTATION SPECIFICATIONS:

- Implement a custom `WritableComparable` type holding two strings.

  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()".

- 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.

- 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.

- Programs to implement and submit (cf. HOMEWORK `hw6` CODE below):
  `StringPairWritable.java`, `WordCoMapper.java`, `SumReducer.java`, `WordCo.java`.

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

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

Provide the last 10 lines of the results file in your written submission.

Problem 2: Secondary Sort (30%)

In this problem we will explore and modify the Secondary Sort problem from Lab 5. 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 5 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?

IMPLEMENTATION SPECIFICATIONS:

- Modify your MAPREDUCE program so that the output is *globally* sorted. The only program that should be updated is the `NameYearPartitioner`.
- Use the same number of Reducers as described in the lab instructions.
- It is sufficient to work with fixed key partitions – no need to use key distributions to avoid skew.
- Programs to implement and submit (cf. HOMEWORK `hw6` CODE below): `NameYearPartitioner.java`.

(c) Now, you want to output the *oldest* person instead of the youngest person. You can do this by merely modifying the job configuration. Do not change the code of any comparators! Describe the change you made to the job configuration and provide the command you used to execute this job.

(d) Why do we need to implement the group comparator in the secondary sort example discussed in Lab 5?

Problem 3: Collaborative Filtering in MAPREDUCE (30%)

The *dual approach* to collaborative filtering is to compute item-item similarites instead of user-user similarites and use those to fill in the missing values in the utility matrix.

(a) Explain two benefits of this approach.

(b) For the input data in the format `<user-id> <movie-id> <rating>` given below, compute the Mapper output, Reducer input and Reducer output for all MAPREDUCE jobs in a **collaborative filtering** approach to compute the **cosine similarity** between **all pairs of movies**. Round to two decimal places.

```
user1 movie1 1
user1 movie3 2
user1 movie2 3
user2 movie2 2
user2 movie3 3
user2 movie5 5
user3 movie1 1
user3 movie2 2
```

continue to next page...

CSE427S

HOMEWORK 6 CODE

M. Neumann

Due: THU 10 OCT 2019 4PM

SUBMISSION INSTRUCTIONS

CODE:

- create a **single zip file** named `hw6_YourName.zip` (or `hw6_YourName_YouPartnersName.zip` for groups) including all files listed below

  Example file names:

  `hw6_MarionNeumann.zip`
  or for teams:
  `hw6_MarionNeumann_AnnaLee.zip`
  or
  `hw6_AnnaLee_MarionNeumann.zip`

- submit the *zip file* to the `hw6` CODE assignment in GRADESCOPE
- your code will be **autograded** for *correctness* and **manually inspected** for *comments*

Problem 4: Custom WritableComparable (20%)

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

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

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

(d) Submit your `WordCo.java` program. 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:

```
StringPairWritable.java
WordCoMapper.java
SumReducer.java
WordCo.java
```

Problem 5: Secondary Sort (10%)

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

Files to be added to your zip file:

```
NameYearPartitioner.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 `hw6_reflection.txt` file provided in the `hw6`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 `hw6` folder and run:

```
$ svn commit -m 'hw6 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_bIRKP1xobp08yIB`.

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!.