Homework 6

CSCIE-55

Fall 2019

Fred Evers

# Homework 6: Hadoop/MapReduce Part I (running MR jobs locally).

Due: Monday, December 2nd, 11:59 Eastern time

Grading: 10%

Last Modified: Monday, 18-Nov-2019 9:39:58 EDT

Unzip the hw\_6.zip.

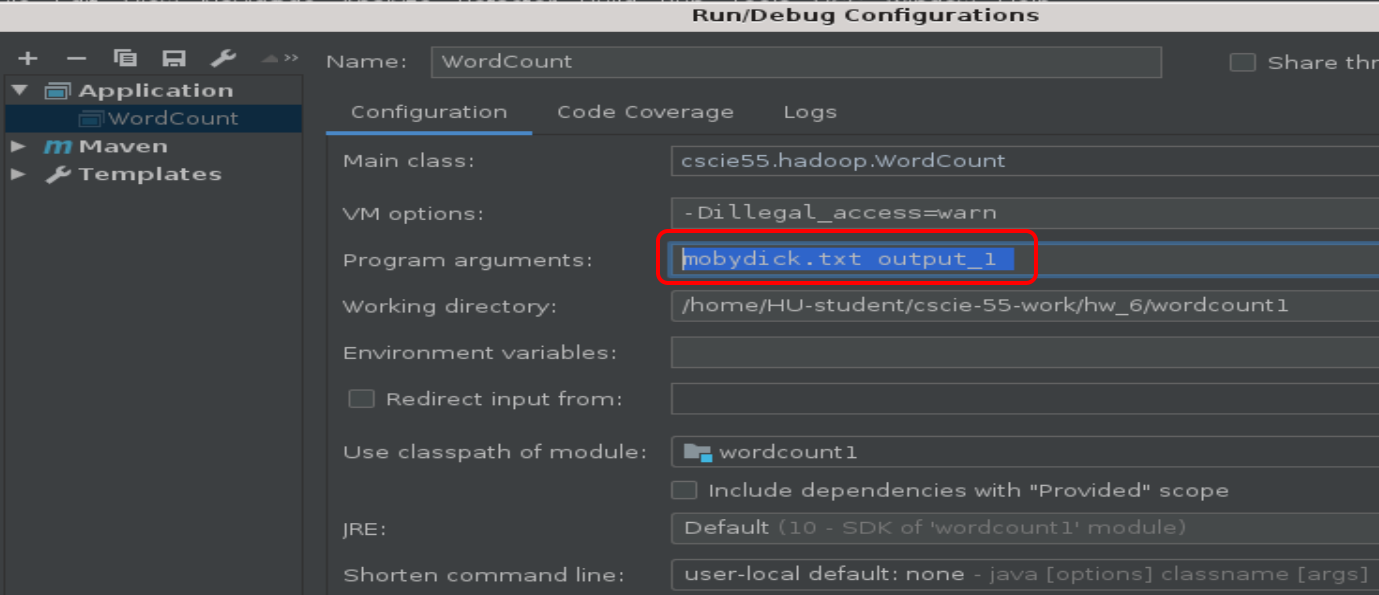
Start Intelli-J, open a new project by clicking Open, browse to the wordcount1.iml file in the unzipped hw\_6/wordcount1 directory, and choose “Open as Project”.

## Problem 1.

Examine the dependencies defined in the pom for wordcount1. Notice that the file modydick.txt is located in the directory ***wordcount1/src/main/resources***

From toolbar, open Run -> Edit Configurations and examine the prepared project named “WordCount”

Notice the ***Program Arguments*** field with 2 string arguments. These are the input parameters to the program. The first specifies the Path [technically the file name] of the input data, the second specifies the name of an output Path [directory].



Notice that on line 56 of WordCount.java, the string that defines the output directory is modified to include ‘target/’ in the output path. This ensures that the output is generated into the target directory and will be deleted any time this project is run under maven, which always calls the ‘clean’ phase before a build, deleting the target folder.

Run the program in Intelli-J.

Congratulations! You have just completed your first Hadoop MapReduce Job!

Expand the target directory in Intelli-J and find the output\_1 directory. There you will find a file named “part-r-0000”. Open this file in Intelli-J and examine it’s contents, Aye?

Now close part-r-0000 and delete the target directory in Intelli-J.

Set a breakpoint on line 28, and run the application in Intelli-j’s debugger to examine what is happening in the MyMapper class’ map() method. [Just do this for a few iterations. There will be many iterations and you don’t need to wait for much. Just see what is happening and how the input ‘Text’ object contains a value that is a line of text, i.e., a String’.]

Open the ‘word’ [Text] object in the debugger’s variable viewer and see how Text contains a String. Note also how the Text.set() method takes a String and creates a Text object.

## Problem 2.

Create a new project in Intelli-J. Call the project “wordcount2”.

Enter ‘cscie55.hw6’ for ‘GroupId’ and ‘wordcount2’ for ‘ArtifactId’. Accept all other defaults in the new project wizard, ensuring that “wordcount2” is the project name.

In the new project, create a new Java class named “WordHistogram.java” , and begin by copy-pasting the WordCount.java file, change class name and any references to “WordCount” to “WordHistogram”. I recommend Ctl-F to use find-replace.

Setup Intelli-j so it accepts input and output params as they were in the provided wordcount1 project.

Select the top-level project and click Run -> Edit configurations. Add a new run config for this project.

In the Program Arguments field, add the same input and output parmeters as found in wordcount1.

Using the problem 2 WordCount2.java from Probal’s section on 11/16 [especially with the data cleaning], modify the map() method in the nested “MyMapper” class to do the following to clean your data while processing it:

* Use String methods to set all words to lowercase.
* Add a regular expression to remove any characters that are not ‘word characters’.
* Trim every word to ensure there is no beginning or trailing white space.

Edit the Mapper and Reducer classes so that the final output of Map/Reduce is a histogram of <word-length, frequency>. For example, assuming you gave “output\_2” as the destination directory, output\_2/part-r-00000 might contain these lines

5 1254

4 6934

…

meaning there were 1,254 words of length 5 and 6,934 words of length 4. Besides making changes to the WordCount1 computation, you will also need to change the parameterized types of your Mapper and Reducer classes from those used by WordCount. As always, make sure the output K,V types of your Mapper match the K,V input types of your Reducer class. Submit the output for analyzing mobydick.txt, as done in the previous problems above.

## Problem 3.

Create another project in Intelli-J. Name this project “seattle\_library”. As before, base the pom on your previous projects. In this case, rather than using mobydick.txt, use file “smaller\_seattle.csv”. In this project, create a new Java class named “SeattleBooks.java”

The data file “smaller\_seattle.csv” contains 3000 lines of comma-separated values describing checkout activity from the Seattle Public Library System. Each entry has the following form:

Physical,Horizon,BOOK,2018,6,1,Bank of the black sheep / Robert Lewis.,"Lewis, Robert","Private investigators Wales Fiction, Cancer Patients Fiction, Amnesia Fiction, Mystery fiction","Serpent's Tail,",2010.

Physical,Horizon,BOOK,2018,6,9,Mat hid / by Lynn Maslen Kertell ; pictures by Sue Hendra.,"Kertell, Lynn Maslen.","Reading Primary Whole word method Juvenile literature, Phonetics Study and teaching Primary Juvenile literature, Vocabulary Juvenile literature, Hide and seek Juvenile fiction","Scholastic,",[2010]

Note that commas separate the values, and the title of the item is given at index 6 (i.e., the 7th position).

Using the methods demonstrated in the ‘SalesCountryStream” demo from section on 11/16, modify the map() method of the MyMapper class so that it splits the input text based on commas. Then clean the entry corresponding to the title. Use Hadoop MapReduce to count the titles to determine the most populate items checked out in Seattle.

Submission requirements:

A zip file containing all 3 Intelli-J projects: wordcount1, wordcount2 and seattle\_library.

Each project should contain:

* All the Java files and the target directories from each problem.
* The output files [part-r-00000] created by running the MR code locally.

A README, explaining each of the MR solutions. [For wordcount1, the ‘solution’ is your observations on what the Text object is.]