CSE 6242 / CX 4242: Data and Visual Analytics | Georgia Tech | Fall 2018

Homework 3 : Hadoop, Spark, Pig and Azure

Prepared by Anmol Chhabria, Arathi Arivayutham, Brendon Duprey, Fan Zhou, Jennifer Ma, Keith Adkins, Kunal Agarwal, Mansi Mathur, Matthew Hull, Matthew Keezer, Michael Petrey, Nilaksh Das, Priyank Madria, Siddharth Gulati, Shruti Shinde, Sneha Venkatachalam, Susanta Routray, Svetlana Afanaseva, Vineet Vinayak Pasupulety, Polo Chau

Submission Instructions and Important Notes:

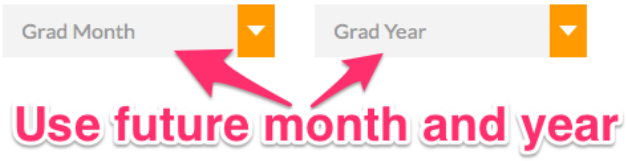
It is important that you read the following instructions carefully and also those about the deliverables at the end of each question or **you may lose points**.

- ❏ Always check to make sure you are using the most up-to-date assignment (version number at bottom right of this document).
- ❏ Submit a single zipped file, called "HW3-{GT username}.zip", containing all the deliverables including source code/scripoints, data files, and readme. Example: "HW3-jdoe3.zip" if GT account username is "jdoe3". Only .zip is allowed (no other format will be accepted). Your GT username is the one with letters and numbers.
- ❏ You may discuss high-level ideas with other students at the "whiteboard" level (e.g., how cross validation works, use hashmap instead of array) and review any relevant materials online. However, each student must write up and submit his or her own answers.
- ❏ All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures (e.g., reported to and directly handled by the Office of Student Integrity (OSI)). Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.
- ❏ At the end of this assignment, we have specified a folder structure about how to organize your files in a single zipped file. 5 points will be deducted for not following this strictly.
- ❏ In your final zip file, do not include any intermediate files you may have generated to work on the task, unless your script is absolutely dependent on it to get the final result (which it ideally should not be).
- ❏ We may use auto-grading scripoints to grade some of your deliverables, so it is extremely important that you strictly follow our requirements.
- ❏ Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
- ❏ Every homework assignment deliverable and every project deliverable comes with a 48-hour "grace period". Any deliverable submitted after the grace period will get zero credit.
- ❏ We will not consider late submission of any missing parts of a homework assignment or project deliverable. To make sure you have submitted everything, download your submitted files to double check.
- ❏ Please make a note that canvas appends a "version number" to files that students resubmit as designed. When a grader downloads the homework, canvas bundles only the latest version of a student's submission. So please make sure that the latest version of (resubmission is intended for grading.

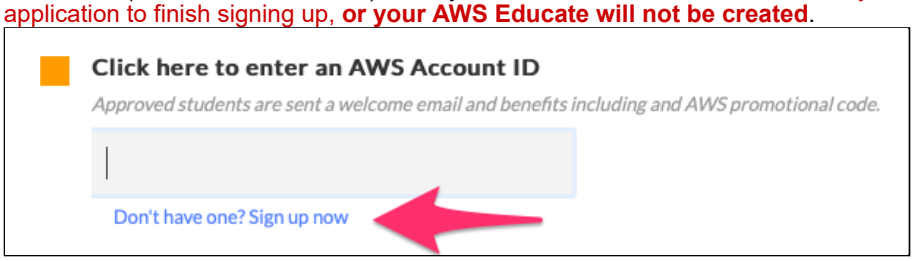
===== DO THIS NOW: apply for AWS Educate account, and set up alarms =====

Apply for an **AWS Educate** account **RIGHT AWAY** to get $100 free AWS credit, and verify that the credit has been properly applied on your account. Creating the account can take days and this assignment's computation can take hours to run, so if you do not do this now, you may not be able to submit your work in time, because you may still be *waiting* for jobs to complete.

- Go to AWS Educate page
- Click the **Join AWS Educate Today** button.
- Choose **Student**, and click **Next**
- Fill out the application
  - You must use your **@gatech.edu** email address (GT is an AWS member school).
  - You must stay with the default option--- **you MUST NOT choose the "starter" option** --- or you will NOT receive the full $100 credit.
  - For "Grad Month" and "Grad Year" (see below), which stand for graduation month and year, **both of them should be in the future**, since you are still a student and haven't graduated yet.



○ You will need to provide the AWS account ID of your AWS account --- **your AWS account is NOT the same as your AWS Educate account**. If you do not have an AWS account yet (thus, no AWS account ID), sign up for one as indicated in the form (see screenshot below). After your AWS account has been created, you MUST return to your AWS Educate application to finish signing up, **or your AWS Educate will not be created**.



○ Your AWS Account ID is at the top of this screen.

● For any emails that you may be receiving from AWS, please remember to check your "spam" email folders. Many students reported emails get pushed to those folders automatically.

EXTREMELY IMPORTANT: enable billing alerts AND set up a billing alarm on AWS to notify you when your credits are running low.

SHUT DOWN EVERYTHING when you're done with the instances (don't leave them on over weekends/holidays!), or you may get surprising credit card bills. Highest record from previous classes went above $2000! **The good news is you can call AWS to explain the situation and they should be able to waive the charges** -- call (phone) AWS customer care **immediately** (not email) to so you can resolve the issue quickly (usually within minutes).

==========

===== DO THIS NOW: get $50 Azure credit =====

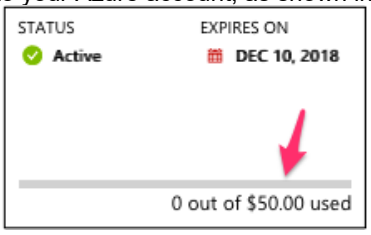
VERY IMPORTANT: Use Firefox or Chrome in incognito/private browsing mode when configuring anything related to Azure (e.g., when using Azure portal), to prevent issues due to browser caches. Safari sometimes loses connections.

Go to https://aka.ms/JoinEdu **RIGHT AWAY** and log in with your **GTUsername@gatech.edu** email address (your GT Username is the one with letters and numbers, e.g., jdoe3). You MUST use this email address, or you may run into issues that we do not know how to solve.

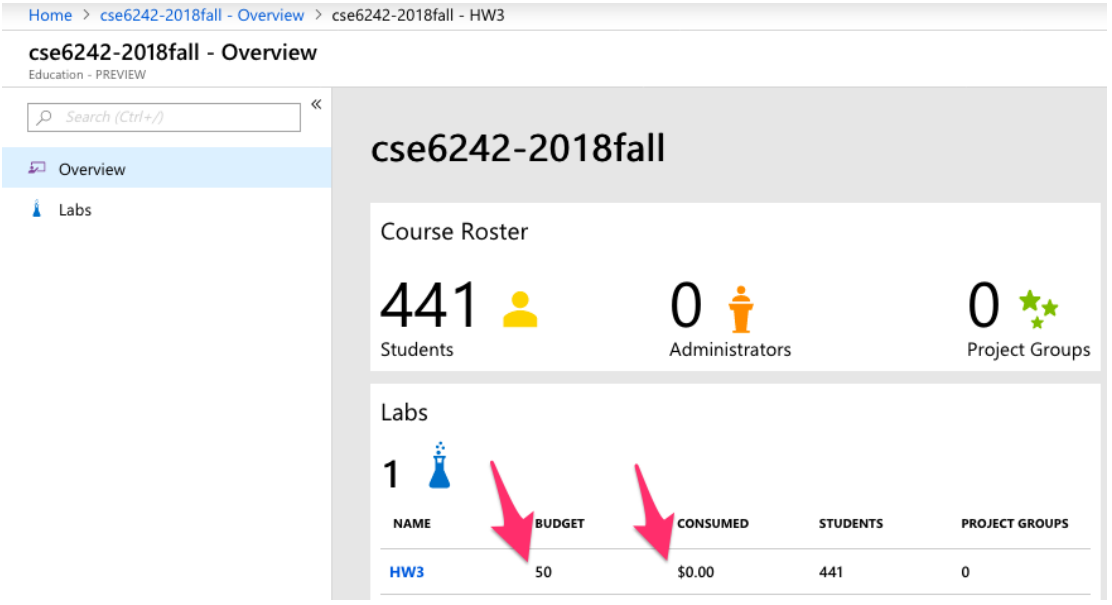
When asked to provide an access code, enter **A32C2**.

If you have NOT created an Azure account before using your GTUsername@gatech.edu email address, one will now be automatically created for you.

We should then see a big blue box titled "You have a new Lab". Click the "Setup Lab" button and complete all steps that follow (some steps require some waiting). Once the setup is complete, click the "Go to Lab" text link, and you should see that $50 has been applied to your Azure account, as shown in the image below:



If you do not see the "Go to Lab" text link, you can use the direct link https://portal.azure.com/#blade/Microsoft_Azure_Education/ClassroomMenuBlade/overview/classroomId/181 to confirm that the credit has been applied, and to track your credit usage, as shown in image below.



You may receive an email titled “Action required: Accept your Azure lab assignment” -- if the $50 has been applied successfully, you can ignore that email.

You can use the long direct link above (portal.azure.com/…181) to check your Azure credit usage as you work on HW3.

The $50 worth of Azure credit should be more than enough for this assignment. If you run out of credit, request extra credit by sending us (instructors) a private Piazza post (tag the post with “azurecredit”), with a brief explanation of why you need the credit. It can take a day or more before you can use the extra credit; if you make last-minute requests, you may not be able to turn in your work in time.

VERY IMPORTANT: Your virtual machine over Azure should only be enabled when you are using Hadoop service. If you are not running any computation over Azure, you should **deallocate your virtual machine** (different from a stopped state) using Azure management portal, to avoid getting charged. Microsoft’s logic here is that you are still occupying some resources over cloud, even if you are not using them for running any computation.

==========

Grading

You can score a maximum of 120 points in this assignment.

Download the HW3 Skeleton before you begin.

Setting up Development Environment for Q1

Installing the Virtual Machine

Download a preconfigured virtual machine (VM) image (~2.7GB). The virtual image comes with pre-installed Hadoop and Java. You will use them for this HW.

Refer to the Instructions to setup and configure the VM. The account details in the VM are:

username: cse6242
password: cse6242

Loading Data into HDFS

Now, let us load our dataset into the HDFS (Hadoop Distributed File System), an abstract file system that stores files on clusters. Your Hadoop code will directly access files on HDFS. Paths on the HDFS look similar to those on the UNIX system, but you cannot explore them directly using standard UNIX commands. Instead, you need to use ***hadoop fs*** commands. For example

```
hadoop fs -ls
```

Download the following two graph files: graph1.zip[1] (~5MB when unzipped) and graph2.zip[2] (~1.1GB when unzipped) into the VM and unzip them. Also place the Q1 skeleton code in the VM. Use the following commands to setup a directory on the HDFS to store the two graph datasets. Do not change the directory structure below (cse6242/) since we will grade your homework using the scripts that assume the following directory structure. First, navigate to the folder in your skeleton code that contains the **src** directory, **pom.xml**, **run1.sh**, and **run2.sh** files and enter the following commands.

```
hadoop fs -mkdir cse6242/
hadoop fs -put path/to/graph1.tsv cse6242
hadoop fs -put path/to/graph2.tsv cse6242
```

Now both files (graph1.tsv and graph2.tsv) are on HDFS, at **cse6242/graph1.tsv** and **cse6242/graph2.tsv**. To check this, try:

```
hadoop fs -ls cse6242
```

Setting up Development Environments

We found that compiling and running Hadoop code can be quite complicated. So, we have prepared some skeleton code, compilation scripts, and execution scripts for you that you can use, in the HW3 skeleton folder. You should use this structure to submit your homework.
In the directory of *Q1*, you will find **pom.xml**, **run1.sh, run2.sh** and the **src** directory.

- The **src** directory contains a main Java file that you will primarily work on. We have provided some code to help you get started. Feel free to edit it and add your files in the directory, but the main class should be Q1. Your code will be evaluated using the provided **run1.sh** and **run2.sh** file (details below).

- **pom.xml** contains the necessary dependencies and compile configurations for each question. To compile, you can simply call Maven in the corresponding directory (i.e., Q1 where pom.xml exists) by this command:

  ```
  mvn package
  ```

  It will generate a single JAR file in the target directory (i.e., target/q1-1.0.jar). Again, we have provided you some necessary configurations to simplify your work for this homework, but you can edit them as long as our run script works and the code can be compiled using mvn package command.

- ***run1.sh***, ***run2.sh*** are the shell script files that run your code over graph1.tsv (run1.sh) or graph2.tsv (run2.sh) and download the output to a local directory. The output files are named based on its question number and graph number (e.g. q1output1.tsv). You can use these run scripts to test your code. Note that these scripts will be used in grading.

  To execute the shell scripts from the command prompt, enter:

  ```
  ./run1.sh
  ./run2.sh
  ```

  Here's what the above scripts do:

  1. Run your JAR on Hadoop specifying the input file on HDFS (the first argument) and output directory on HDFS (the second argument)
  2. Merge outputs from output directory and download to local file system.
  3. Remove the output directory on HDFS.

Q1 [15 points] Analyzing a Graph with Hadoop/Java

Imagine that your boss gives you a large dataset which contains an entire email communication network from a popular social network site. The network is organized as a directed graph where each node represents an email address and the edge between two nodes (e.g., Address A and Address B) has a weight stating how many times A wrote to B. You have been tasked with finding which people have sent the most emails. Your task is to write a MapReduce program in Java to report, for each node in the graph, the largest weight among all of the node's weighted outbound edges.

First, go over the Hadoop word count tutorial to familiarize yourself with Hadoop and some Java basics. You will be able to complete this question with only some knowledge about Java. You should have already loaded two graph files into HDFS and loaded into your HDFS file system in your VM. Each file stores a list of edges as tab-separated-values. Each line represents a single edge consisting of three columns: (source node ID, target node ID, edge weight), each of which is separated by a tab (\t). Node IDs and weights are nonnegative integers. Below is a small toy graph, for illustration purposes (on your screen, the text may appear out of alignment).

```
src         tgt          weight
110         10          3
200         10          1
150         200          30
110         100          10
200         110          15
110         130          67
```

Your program should not assume the edges to be sorted or ordered in any ways (i.e., your program should work even when the edge ordering is random).

Your code should accept two arguments upon running. The first argument (*args[0]*) will be a path for the input graph file on HDFS (e.g., cse6242/graph1.tsv), and the second argument (*args[1]*) will be a path for output directory on HDFS (e.g., cse6242/q1output1). The default output mechanism of Hadoop will create multiple files on the output directory such as part-00000, part-00001, which will be merged and downloaded to a local directory by the supplied run script. Please use the run1.sh and run2.sh scripts for your convenience.

The format of the output should be such that each line represents a node ID and the largest weight among all its outbound edges. The ID and the largest weight must be separated by a tab (\t). Lines do not need to be sorted. The following example result is computed based on the toy graph above. Please exclude nodes that do not have outgoing edges (e.g., those email addresses which have not sent any communication).

For the toy graph above, the output is as follows.

```
110         67
200         15
150         30
```

Deliverables

1. **[5 points] Your Maven project directory including Q1.java.** Please see detailed submission guide at the end of this document. You should implement your own MapReduce procedure and should not import external graph processing library.

2. **[5 points] q1output1.tsv**: the output file of processing graph1.tsv by run1.sh.

3. **[5 points] q1output2.tsv**: the output file of processing graph2.tsv by run2.sh.

Q2 [25 pts] Analyzing a Large Graph with Spark/Scala on Databricks

Tutorial: First, go over this Spark word count tutorial to get more background about Spark/Scala.

Goal

Your objectives:

1. Eliminate any duplicate rows.
2. Filter the graph such that only nodes containing an edge weight >= 5 are preserved.
3. Analyze the graph to find the nodes with the highest in-degree, out-degree, and total degree using DataFrame operations.
4. Download a new DataFrame to **q2output.csv** containing your analysis (schema provided below).

You will analyze bitcoinalpha.csv[3] using Spark and Scala on the Databricks platform. This graph is a who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin Alpha. For a node, the number of incoming edges is the in-degree of the node and the number of outgoing edges is called the out-degree. The total degree is the sum of all edges for a node.

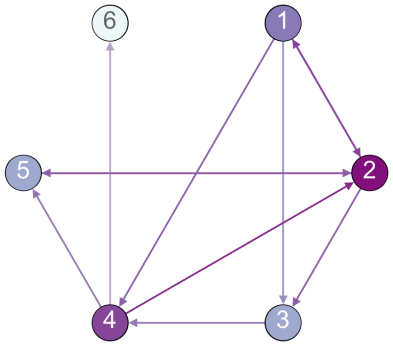
You should perform this task using the DataFrame API in Spark. Here is a guide that will help you get started on working with data frames in Spark.

A template Scala notebook, q2-skeleton.dbc has been included in the HW3-Skeleton that reads in a sample graph file toygraph.csv. In the template, the input data is loaded into a dataframe, inferring the schema using reflection (Refer to the guide above).

Note: You must use only Scala DataFrame operations for this task. You will lose points if you use SQL queries, Python, or R to manipulate a dataframe.

You may find some of the following DataFrame operations helpful:
`toDF, join, select, groupBy, orderBy`

Upload the data file toygraph.csv and q2-skeleton.dbc to your Databricks workspace before continuing. Follow the Databricks Setup Guide for further instructions. Consider the following directed graph example and how to accomplish the stated objectives



src	tgt	weight
1	2	5
1	3	5
1	4	5
2	1	5
2	3	5
2	5	5
3	4	5
4	2	5
4	5	5
4	6	5
5	2	5
1	2	5
3	1	4

1. Eliminate Duplicates: The second instance of `src: 1 - tgt: 2` should be eliminated from graph snippet below.

src	tgt	weight
1	2	5
1	3	5
.	.	.
.	.	.
~~1~~	~~2~~	~~5~~
3	1	4

2. Filter the graph such that only nodes containing an edge weight >= 5 are preserved
3. Find node w/ highest in-degree, out-degree, and highest total degree. If we analyzed the toy graph, we would find the following:

node	out-degree	in-degree	total-degree
1	3	1	4
2	3	3	6
3	1	2	3
4	3	2	5
5	1	2	3
6	0	1	1

Nodes(s) with the highest in-degree : **2**
Node(s) with the highest out-degree: **1**, **2, 4**
Node(s) with highest combined degree: **2**

Notes
If two or more nodes have the same **out-degree**, report the one with the lowest node id
If two or more nodes have the same **in-degree**, report the one with the lowest node id
If two or more nodes have the same **total degree**, report the one with the lowest node id

4. Create a dataframe to store your results using this schema:

3 columns, named: 'v', 'd', 'c' where:

- **v** : vertex id

- `d` : degree calculation (an integer value. one row with highest in-degree, a row w/ highest out-degree, a row w/ highest total degree)

- `c` : category of degree, containing one of three string values:
  'i' : in-degree
  'o' : out-degree
  't' : total-degree

Your output will be downloaded as a .csv file that meets the following requirements:

1. Your output shall contain exactly 4 rows. (1 header row + 3 data rows)
2. Your output shall contain exactly the column order specified.
3. The order of rows does not matter.

A correct output .csv for the input file toygraph.csv would look like:

```
v,d,c
2,3,i
1,3,o
2,6,t
```

Whereas:
Node 1 has highest out-degree with a value of 3
Node 2 has highest in-degree with a value of 3
Node 2 has highest total degree with a value of 6

Deliverables

1. **[10 pts]**
   a. **q2.dbc** Your solution as Scala Notebook archive file (.dbc) exported from Databricks. See the Databricks Setup Guide on creating an exportable archive for details.
   b. **q2.scala**, Your solution as a Scala source file exported from Databricks. See the Databricks Setup Guide on creating an exportable source file for details.

   Notes: you are exporting your solution as both a .dbc & a .scala file.

2. **[15 pts] q2output.csv:** The output file of processing **bitcoinalpha.csv** from the q2 notebook file.

Q3 [35 points] Analyzing Large Amount of Data with Pig on AWS

You will try out Apache Pig for processing n-gram data on Amazon Web Services (AWS). This is a fairly simple task, and in practice you may be able to tackle this using commodity computers (e.g., consumer-grade laptops or desktops). However, we would like you to use this exercise to learn and solve it using distributed computing on Amazon EC2, and gain experience (very helpful for your career), so you will be prepared to tackle problems that are more complex.

The services you will primarily be using are Amazon S3 storage, Amazon Elastic Cloud Computing (EC2) virtual servers in the cloud, and Amazon Elastic MapReduce (EMR) managed Hadoop framework.

For this question, you will only use up a very small fraction of your $100 credit. AWS allows you to use up to 20 instances in total (that means 1 master instance and up to 19 core instances) without filling out a "limit request form". For this assignment, you should not exceed this quota of 20 instances**.** Refer to details about instance types, their specs, and pricing. In the future, for larger jobs, you may want to use AWS's pricing calculator.

AWS Guidelines

Please read the AWS Setup Guidelines provided to set up your AWS account.

Datasets

In this question, you will use subsets of the Google books *n-grams* dataset (full dataset for reference), on which you will perform some analysis. An '*n*-gram' is a phrase with *n* words; the full n-gram dataset lists n-grams present in the books on books.google.com along with some statistics.

You will perform your analysis on two custom datasets, extracted from the Google books bigrams (2-grams), that we have prepared for you: a small one **s3://cse6242oan-2018fall-aws-small** (~1GB) and a large one **s3://cse6242oan-2018fall-aws-big** (~150GB).

VERY IMPORTANT: Both the datasets are in the **US East (N. Virginia)** region. Using machines in other regions for computation would incur data transfer charges. Hence, set your region to **US East (N. Virginia)** in the beginning (not Oregon, which is the default). **This is extremely important otherwise your code may not work and you may be charged extra.**

The files in these two S3 buckets are stored in a tab ('\t') separated format. Each line is in the following format:

```
n-gram TAB year TAB occurrences TAB books NEWLINE
```

Some example lines:

```
I am        1936        342         90
I am        1945        211         25
I am        1951        47         12
very cool        1923        118         10
very cool        1980        320         100
very cool        2012        994         302
very cool        2017        1820         612
```

The above lines tell us that, in 1936, the bigram "I am" appeared 342 times in 90 different books. In 1945, "I am" appeared 211 times in 25 different books. And so on.

Goal

Output the 15 bigrams having the highest **average number of appearances per book** along with their corresponding averages, in **tab-separated format**, sorted in descending order. Only consider entries with at least 300 occurrences and at least 12 books. If multiple bigrams have the same average, **order them alphabetically**. For the example above, the output will be:

```
I am        3.80
very cool        3.09
```

Refer to the calculations given below

```
I am                (342) / (90) = 3.80
very cool        (320 + 994 + 1820) / (100 + 302 + 612) = 3.09
```

Sample Output

To help you evaluate the correctness of your output, we provide you with the output for the small dataset.

Note: Please strictly follow the formatting requirements for your output as shown in the small dataset output file. You can use https://www.diffchecker.com/ to make sure the formatting is correct. Improperly formatting outputs may not receive any points.

Using PIG (Read these instructions carefully)

There are two ways to debug PIG on AWS (all instructions are in the AWS Setup Guidelines):

1. **Use the interactive PIG shell** provided by EMR to perform this task from the command line (grunt). Refer to Section 8: Debugging in the AWS Setup Guidelines for a detailed step-by-step procedure. You should use this method if you are using PIG for the first time as it is easier to debug your code. However, as you need to have a persistent ssh connection to your cluster until your task is complete, this is suitable only for the smaller dataset.

2. **Upload a PIG script** with all the commands which computes and direct the output from the command line into a separate file. Once you verify the output on the smaller dataset, use this method for the larger dataset. You don't have to ssh or stay logged into your account. You can start your EMR job, and come back after a few hours when the job is complete!

Note: In summary, verify the output for the smaller dataset with Method 1 and submit the results for the bigger dataset using Method 2.

Sample Commands: Load data in PIG

To load the data from the **s3://cse6242oan-2018fall-aws-small** bucket into a PIG table, you can use the following command:

```
grunt> bigrams = LOAD 's3://cse6242oan-2018fall-aws-small/*' AS (bigram:chararray, year:int, occurrences:int, books:int);
```

Note:

- Refer to other commands such as LOAD, USING PigStorage, FILTER, GROUP, ORDER BY, FOREACH, GENERATE, LIMIT, STORE, etc.
- Copying the above commands directly from the PDF and pasting on console/script file may lead to script failures due to the stray characters and spaces from the PDF file.
- Your script will fail if your output directory already exists. For instance, if you run a job with the output folder as **s3://cse6242oan-<GT account username>/output-small**, the next job which you run with the same output folder will fail. Hence, please use a different folder for the output for every run.
- You might also want to change the input data type for **occurrences** and **books** to handle floating point values.
- While working with the interactive shell (or otherwise), you should first test on a small subset of the data instead of the whole data (the whole data is over 100 GB). Once you believe your PIG commands are working as desired, you can use them on the complete data and wait since it will take some time.

Deliverables

- **pig-script.txt**: The PIG script for the question (using the **larger** data set).
- **pig-output.txt**: Output **(tab-separated)** (using the **larger** data set).

Note: Please strictly follow the guidelines below, otherwise your solution may not be graded.

- Ensure that file names (case sensitive) are correct.
- Ensure file extensions (.txt) are correct.
- The size of each pig-script.txt and pig-output.txt file should not exceed 5 KB.
- Double check that you are submitting the correct set of files --- we only want the script and output from the larger dataset. Also double check that you are writing the right dataset's output to the right file.
- Ensure that unnecessary new lines, brackets, commas etc. aren't in the file.
- Please use tabs (not space) in the output file for separating the 2 columns.

Q4 [35 points] Analyzing a Large Graph using Hadoop on Microsoft Azure

VERY IMPORTANT: Use Firefox or Chrome in incognito/private browsing mode when configuring anything related to Azure (e.g., when using Azure portal), to prevent issues due to browser caches. Safari sometimes loses connections.

Goal

The goal is to analyze graph using Microsoft Azure, and your task is to write a MapReduce program to compute the distribution of a graph's node degree differences (see example below). Note that this question shares some similarities with Question 1 (e.g., both are analyzing graphs). Question 1 can be completed using your own computer. This question is to be completed using Azure. We recommend that you first complete Question 1.

You will use two data files in this questions:

- small.tsv[4] (zipped as 10MB small.zip; ~38MB when unzipped)
- large.tsv[5] (zipped as 900MB large.zip; ~3GB when unzipped)

Each file stores a list of edges as tab-separated-values. Each line represents a single edge consisting of two columns: (Source, Target), each of which is separated by a tab. Node IDs are positive integers and the rows are already sorted by Source.

```
Source	Target
1	2
2	1
2	3
3	2
4	2
4	3
```

Your code should accept two arguments upon running. The first argument (args[0]) will be a path for the input graph file, and the second argument (args[1]) will be a path for output directory. The default output mechanism of Hadoop will create multiple files on the output directory such as part-00000, part-00001, which will have to be merged and downloaded to a local directory.

The format of the output should be as follows. Each line of your output is of the format

```
diff	count
```

where
(1) `diff` is the difference between a node's out-degree and in-degree (out-degree - in-degree); and
(2) `count` is the number of nodes that have the value of `difference` (specified in 1).
The out-degree of a node is the number of edges where that node is the Source. The in-degree of a node is the number of edges where that node is the Target. `diff` and `count` must be separated by a tab (\t), and lines do not have to be sorted.

The following result is computed based on the toy graph above.

```
-1          2
0           1
2           1
```

The explanation of the above example result:

Output	Explanation
`-1 2`	There are 2 nodes (node 2 and 3) whose degree difference is -1
`0 1`	There is 1 node (node 1) whose degree is 0
`2 1`	There is 1 node (node 4) whose degree difference is 2

Hint: One way of doing it is using the mapreduce procedure twice. The first one for finding the difference between out-degree and in-degree for each node, the second for calculating the node count of each degree difference. You will have to make changes in the skeleton code for this.

In the Q4 folder of the hw3-skeleton, you will find the following files we have prepared for you:

- ***src*** directory contains a main Java file that you will work on. We have provided some code to help you get started. Feel free to edit it and add your files in the directory, but the main class should be called "Q4".
- ***pom.xml*** contains necessary dependencies and compile configurations for the question.

To compile, you can run the command:

```
mvn clean package
```

in the directory which contains pom.xml.

This command will generate a single JAR file in the target directory (i.e. target/q4-1.0.jar).

Creating Clusters in HDInsight using the Azure portal

Azure HDInsight is an Apache Hadoop distribution. This means that it handles large amount of data on demand. The next step is to use Azure's web-based management tool to create a Linux cluster. Follow the documentation here to create a new cluster -- make sure to use the following settings

- Select "Quick Create" instead of "Custom"
- "Subscription" drop down menu: select "Microsoft Azure Sponsorship 2"
- "Cluster type": choose "Hadoop 2.7.3 (HDI 3.6)"

At the end of this process, you will have created and provisioned a New HDInsight Cluster and Storage (the provisioning will take some time depending on how many nodes you chose to create). Please record the following important information for later use:

- Cluster login credentials
- SSH credentials
- Container credentials

VERY IMPORTANT: HDInsight cluster billing starts once a cluster is created and stops when the cluster is deleted. To save the credit, you'd better to delete your cluster when it is no longer in use. Please refer https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-delete-cluster.

Uploading data files to HDFS-compatible Azure Blob storage

We have listed the main steps from the documentation for uploading data files to your Azure Blob storage here:

1. Install Azure CLI.

2. Open a command prompt, bash, or other shell, and use `az login` command to authenticate to your Azure subscription. When prompted, enter the username and password for your subscription.

3. `az storage account list` command will list the storage accounts for your subscription.
4. `az storage account keys list --account-name <storage-account-name> --resource-group <resource-group-name>` command should return Primary and Secondary keys. Copy the Primary key value because it will be used in the next steps.
5. `az storage container list --account-name <storage-account-name> --account-key <primary-key-value>` command will list your blob containers.

6. `az storage blob upload --account-name <storage account name> --account-key <primary-key-value> --file <small or large .tsv> --container-name <container-name> --name <name for the new blob>/<small or large .tsv>` command will upload the source file to your blob storage container.

Using these steps, upload small.tsv and large.tsv to your blob storage container. After that write your hadoop code locally and convert it to a jar file using the steps mentioned above.

Uploading your Jar file to HDFS-compatible Azure Blob storage

Azure Blob storage is a general-purpose storage solution that integrates with HDInsight. Your Hadoop code should directly access files on the Azure Blob storage.

Upload the jar file created in the first step to Azure storage using the following command:

```
scp <your-relative-path>/q4-1.0.jar <ssh-username>@<cluster-name>-ssh.azurehdinsight.net:
```

SSH into the cluster using the following command:

```
ssh <ssh-username>@<cluster-name>-ssh.azurehdinsight.net
```

Note: if you see the warning - REMOTE HOST IDENTIFICATION HAS CHANGED, you may clean /home/<user>/.ssh/known_hosts". Refer to host identification.

Run the ls command to make sure that the `q4-1.0.jar` file is present.
To run your code on the small.tsv file, run the following command:

```
yarn jar q4-1.0.jar edu.gatech.cse6242.Q4 wasbs://<container-name>@<blob-storage-name>.blob.core.windows.net/<small-blob-name>/small.tsv wasbs://<container-name>@<blob-storage-name>.blob.core.windows.net/smalloutput

Command format: yarn jar jarFile packageName.ClassName dataFileLocation outputDirLocation
```

Note: if "`Exception in thread "main" org.apache.hadoop.mapred.FileAlreadyExistsException...`" occurs, you need to delete the output folder from your Blob. You can do this at portal.azure.com.

The output will be located in the `wasbs://<container-name>@<blob-storage-name>.blob.core.windows.net/smalloutput`. If there are multiple output files, merge the files in this directory using the following command:

```
hdfs dfs -cat wasbs://<container-name>@<blob-storage-name>.blob.core.windows.net/smalloutput/* > small.out

Command format: hdfs dfs -cat location/* >outputFile
```

Exit to your local machine:

```
exit
```

Download the merged file to the local machine (this can be done either from https://portal.azure.com/ or by using the scp command from the local machine). Here is the scp command for downloading this output file to your local machine:

```
scp <ssh-username>@<cluster-name>-ssh.azurehdinsight.net:/home/<ssh-username>/small.out .
```

Using the above command from your local machine will download the small.out file into the current directory. Repeat this process for large.tsv.

Deliverables

1. [15pt] **Q4.java & q4-1.0.jar:** Your java code and converted jar file. You should implement your own map/reduce procedure and should not import external graph processing library.
2. [10pt] **small.out**: the output file generated after processing small.tsv.
3. [10pt] **large.out**: the output file generated after processing large.tsv.

Q5 [10 points] Regression: Automobile price prediction, using Azure ML Studio

Note: Create and use a free workspace instance at https://studio.azureml.net/ instead of your Azure credit for this question.

Goal

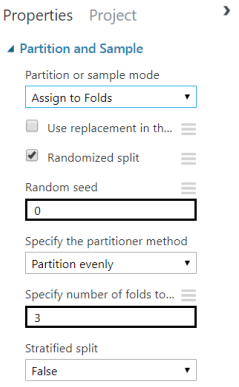
This question's main purpose is to introduce you to Microsoft Azure Machine Learning Studio and familiarize you with its basic functionality and typical machine learning workflow. Go through the "Automobile price prediction" tutorial and complete the tasks below.

You will modify the given file, **results.csv**, by adding your results for each of the tasks below. We will autograde your solution, therefore DO NOT change the order of the questions or anything else. Report the exact numbers that you get in your output, DO NOT round the numbers.

1. [3 points] Repeat the experiment mentioned in the tutorial and report the values of the metrics as mentioned in the '*Evaluate Model'* section of the tutorial.
2. [3 points] Repeat the same experiment, change the '*Fraction of rows in the first output*' value in the split module to 0.85 (originally set to 0.75) and report the corresponding values of the metrics.
3. [4 points] Evaluate the model with the 3-fold cross-validation (CV), select the parameters in the module *'Partition and sample'* (Partition and Sample) (see figure below). Report the value of Root Mean Squared Error (RMSE) for each fold.

Specifically, you need to do the following:

A. Create a new model (Linear Regression)
B. Import the entire dataset (Automobile Price Data (Raw))
C. Clean the missing data by removing rows that have any missing values
D. Perform cross validation on the dataset obtained after Step C



Deliverables

1. [10pt] **results.csv:** a csv file containing results for all of the three parts.

Important: folder structure of the zip file that you submit

You are submitting a single zip file HW3-GTUsername.zip (e.g., HW3-jdoe3.zip, where "jdoe3" is your GT username), which must unzip to the following directory structure (i.e., a folder "HW3-jdoe3", containing folders "Q1", "Q2", etc.). The files to be included in each question's folder have been clearly specified at the end of each question's problem description above.

```
HW3-GTUsername/
    Q1/
        src/main/java/edu/gatech/cse6242/Q1.java
        pom.xml
        run1.sh
        run2.sh
        q1output1.tsv
        q1output2.tsv
        (do not attach target directory)
    Q2/
        q2.dbc
        q2.scala
        q2output.csv
    Q3/
        pig-script.txt
        pig-output.txt
    Q4/
          src/main/java/edu/gatech/cse6242/Q4.java
          pom.xml
          q4-1.0.jar (from target directory)
          small.out
          large.out
```

(do not attach target directory)

Q5/

results.csv

Version 5

---

[1] Graph derived from the LiveJournal social network dataset, with around 30K nodes and 320K edges.
[2] Graph derived from the LiveJournal social network dataset, with around 300K nodes and 69M edges.
[3] Graph derived from the Stanford Large Network Dataset Collection
[4] subset of Youtube social network data
[5] subset of Friendster data