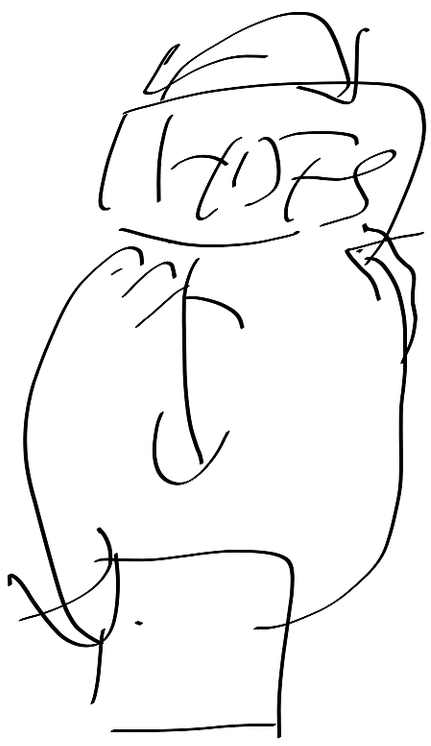
CSE 482: Big Data Analysis (Fall 2017) Homework 4
Due date: Monday, December 12, 2017

Please make sure you submit the homework via the handin system.

1. Write the corresponding HDFS commands to perform the following tasks. Each of these tasks must be accomplished with a single HDFS command. Hint: type `hadoop fs -help` for the list of commands available. Note the difference between the local (Linux) file directory and HDFS directory. To double-check your answers, you're encouraged to test the commands (on your own HDFS directory) to make sure they work correctly.



   (a) Make a directory on HDFS named `/user/hadoop`. ⇒ hadoop fs -mkdir /user/hadoop
   (b) Upload a file named `data.txt` from the local filesystem to HDFS. hadoop fs -copyfromLocal data.txt
   (c) Rename the uploaded file on HDFS from `data.txt` to `temp.txt` hadoop fs -mv data.txt temp.txt
   (d) Move the file from its current location at `/user/hadoop/temp.txt` to `/user/hduser/temp.txt`. hadoop fs -mv /user/hadoop/temp.txt /user/hduser/temp.txt
   (e) Copy the file from `/user/hduser/temp.txt` to `/user/hadoop/data.txt` (note that the filename is also changed). hdfs fs -cp /user/hduser temp.txt /user/hadoop/data
   (f) Display the content of the file `/user/hduser/temp.txt`. ⇒ hadoop fs -cat /user/hduser/temp.txt
   (g) Delete the file `/user/hadoop/data.txt`. ⇒ hadoop fs -rm /user/hadoop/data.txt
   (h) Download the file named `/user/hduser/temp.txt` to the local filesystem directory `/user/cse482`. ⇒ hadoop dfs -copyTo local /user/hduse/temp.xt /user/cse482
   (i) Merge all the files under the HDFS directory `/user/hadoop/results` into a single file `output.txt` to be stored in your current working directory. ⇒ hadoop fs -getmerge /user/hadoop/results output.txt
   (j) Changing permission of the HDFS ~~dir~~ectory `/user/hduser` so that only the user hduser can read and write files stored in the directory (but not other users).
   hadoop fs chmod -R 600 /user/hd suse

b)

- mapper function:
  tokeniz each user and set first law of first user is fixed user and lease of user list are friends of fixed user. Then make key pair of friendship value 1 then make another pair with fixed friend users for friend of friend.

  Ex) A B C D ⇒ {(A,B) ⇒ 1}
  fixed user / friend of fixed user
  {(A,C) ⇒ 1}
  ⋮
  ⇒ {(B,C) ⇒ 0}
  {(C,D) ⇒ 0}
  ⋮

- mapper output: key is tuple of friends relationship value is 1 and another tuple of friend of friend relationship value is 0.

- Reducer input: all list of tuple, value 1 and 0

- Reducer function: check the tuple is duplicate and if it is duplicate then drop. Do not chose 1 value tuple because they are friend relationship. Then select only 0 value tuples.

- Reducer output:
  tuple(fixedU, fofU) where fixedU < fofU, key is fixedU
  value: fofU
  ("0" value tuple was friend of friend relationship)

2. For each problem and data set described below, state how would you setup the (key,value) pairs as inputs and outputs for the mappers and reducers. Also, explain the operation performed by the map and reduce functions given their input. Assume your Hadoop program uses TextInputFormat as its input format (where each record corresponds to a line of the input file). Since the inputs for the mappers are the same (byte offset, content of the line), you only need to specify the mappers' outputs. If the problem requires more than one mapreduce jobs, specify the role of each map/reduce function along with their input and output key-value pairs). You should solve the problem with minimum number of mapreduce jobs.

Example:

Data set: Collections of text documents.

Problem: Count the frequency of nouns that appear at least 100 times in the documents.

Answer:

Mapper function: Tokenize each line into a set of terms (words), and filter out terms that are not nouns.

Mapper output: key is a noun, value is 1.

Reducer input: key is a word, value is list of 1's.

Reduce function: sums up the 1's for each key (noun).

Reducer output: key is a noun, value is frequency of the word (filter the nouns whose frequencies are below 100).

{S1 → {0,0}
S1 → {0,0}
S1 → {1,0}
S1 → {1,1}

a)

- mapper function:
  tokenize each key information such as A123456074 / CSE260 / FS2013 then set key and value
- Mapper output: key is student Id, Value: Tuple type {credits, GPA}
- Reducer input: key is student ID, value is list of tuple {credits, GPA} set
- Reduce function: multiply credits and GPA in each tuple and add up the multiply numbers, and add up just credits columns and devided multiply add-up number.
- Reduce output
  Key = student Id, Value = GPA Average.

(a) **Data set:** Student transcript database. Each record consists the following information: student ID, course code, semester enrolled, number of credits, and GPA for the course. For example:

A123456074 CSE260 FS2013 3 3.5

GPA is a numeric-valued attribute ranging from 0.0 to 4.0.

Problem: Compute the average GPA for each student.

(b) **Data set:** Facebook friendship graph. Each record corresponds to a Facebook user, followed by the list of his/her friends. For example, the graph data may contain the following records:

john123 mary456 tom312 lee222
mary456 john123
tom312 john123 lee222
lee222 john123 tom312 → 2

Problem: Find pairs of users who are not friends with each other but share one or more common friends. This is known in network science literature as the "friend-of-a-friend" (FOF) problem. For example, mary456 and tom312 share a common friend, john123, but they are not friends with each other. In this case, the hadoop program should output the pair (mary456, tom312) only once, i.e., it should not output the duplicate pair (tom312, mary456).

2 ( ) → 1
( ) → 0

b)

c)

- mapper function: get each line and mapping the set with user_id and make tuple artist and vote
- mapper output: key is user_id and value is tuple (artist, vote)
- reducer input: key is user_id and value is list of tuple (artist, vote)
- reducer function: select only 1 value from tuple[vote] position for each user
- reduce output: key: user_Id value: tuple (favorite artist, highest vote)

d)

- mapper function: Get each line and mapping with each station_id and tuple with date, total_precipitation
- mapper output: key: station_Id value: tuple(date, total_precipitation)
- reducer Input: key: station_Id value: list of tuple(date, total_precipitation)
- reducer function: Get mean value and standard deviations from each station set. Then using those values to calculate z-score. And if z-score is smaller than −3 or bigger than 3 means not normal. So drop the not normal value and also drop the negative total_precipitation value set.
- reducer output: key: station_Id value: date(filtered normal z-score and not negative total-precipitation value)

(c) **Data set:** Online music streaming data. Each line in the data file has 4 columns (user_id, song_title, artist, vote), where vote is an integer value with the following values: +1 (the user liked the song), 0 (the user did not rate the song), -1 (the user disliked the song).
Problem: For each user, list his/her favorite artist, i.e., the artist that received the highest net vote. Do not output anything if the net vote is non-positive (0 or negative).

(d) **Data set:** Measurements of total daily precipitation for weather stations from around the world. Each line in the data files has 3 columns (station_id, date, total_precipitation).
Problem: Find the station_id and date of anomalous precipitation values. A precipitation value is considered anomalous if it is negative-valued or is 3 standard deviations or more away from its mean.

(e) **Data set:** Online 2-player gaming data. Each line in the data file has the following information:

```
gameID player1 player2 player1Score player2Score
```

For example, the line

```
123311 Wchampion lansing233 50 24
```

suggests that `Wchampion` won the game by a score of 50 to 24 (the winner is the player who achieves a score of 50 or higher). Note that a player can play against the same opponent more than once. Assume each player in the dataset has played more than 100 games.
Problem: For each player, find the toughest opponent to play against (i.e., the opponent to whom they have lowest winning percentage). For example, if `Wchampion` has a winning percentage of only 0.20 when playing against `msu4life` and has a winning percentage higher than 0.2 when playing against other opponents, the program should output the pair (playerID, toughestOpponent):

```
Wchampion, msu4life
```

For tie-breaking cases (i.e., if a player has more than one toughest opponent with lowest winning percentage), the program may randomly select one of the toughest opponents as its output. If a player has won all of his/her games, then the program should output the pair (playerID, NoToughestOpponent).

e)
- mapper function: Get each line for games, select player1, player2 (opponents), player1 score, and player2 score, then Compare player1 score and player2 score, then if player1 score is bigger than player2 score set win value "1", if is same or smaller then set win value as "0". Mapping with player1 and tuple of player2 and win value(0 or 1)
- mapper output: key player1 and values: tuple (player2, win value)
- reducer Input: key: player1 value: list of tuple (player2, win value)
- reducer function: Count that how many game player1 done, and then add up all win value(0 or 1), it tell how many game player1 win from game. Then Calculate wining percentage base on these value. Then Compare each player's winning percentage and then sort order by the percentage and make pair with own player and higher winning percentage player, if it is same then randomly make pair.
- reducer output key: player1_ID value: The ID for toughest opponents