Aims

This exercise aims to get you to practice more on Spark Programming.

Background

The transformation and action functions examples are available at:

http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html

The GraphX programming guide is at:

https://spark.apache.org/docs/latest/graphx-programming-guide.html

Scala string interpolation methods (used to format your output) can be found at:

https://docs.scala-lang.org/overviews/core/string-interpolation.html

A tutorial of Scala is available at:

http://docs.scala-lang.org/tutorials/?_ga=1.99469143.850382266.1473265612

Spark core programming

Question 1. Download the input text file pg100.txt from: https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/15854. Compute the average length of words starting with each letter. This means that for every letter, you need to compute: the total length of all words that start with that letter divided by the total number of words that start with that letter.

- Ignore the letter case, i.e., consider all words as lower case.
- Ignore terms starting with non-alphabetical characters, i.e., only consider terms starting with "a" to "z".
- The length of a term X can be obtained by X.length.
- Use the following split function to split the documents into terms:

```
split("[\\s*$&#/\"'\\,.:;?!\\[\\](){}<>~\\-_]+")
```

Your Spark program should generate a list of key-value pairs. Keys and values are separated by ",", and the values are of double precision, ranked in alphabetical order. You can see the result at: https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/15013.

Name your scala file as “Problem1.scala”, the object as “Problem1”, and put it in a package “comp9313.lab8”. Put the input file in HDFS folder “/user/comp9313/input”, and store your output in HDFS folder “user/comp9313/output”. The input and output paths are obtained from the arguments.

Question 2. Download the sample input file “Votes.csv” from: https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/15015, and put it in HDFS folder “/user/comp9313/input”. In this file, the fields are separated by ‘,’ and the lines are separated by ‘\n’. The data format of “Votes.csv” is as below:

```
      - Id
      - PostId
      - VoteTypeId
         - ` 1`: AcceptedByOriginator
         - ` 2`: UpMod
         - ` 3`: DownMod
         - ` 4`: Offensive
         - ` 5`: Favorite - if VoteTypeId = 5 UserId will be populated
         - ` 6`: Close
         - ` 7`: Reopen
         - ` 8`: BountyStart
         - ` 9`: BountyClose
         - `10`: Deletion
         - `11`: Undeletion
         - `12`: Spam
         - `13`: InformModerator
         - `14`:
         - `15`:
         - `16`:
      - UserId (only for VoteTypeId 5)
      - CreationDate
```

(i). Find the top-5 VoteTypeIds that have the most distinct posts. You need to output the VoteTypeId and the number of posts. The results are ranked in descending order according to the number of posts, and each line is in format of: VoteTypeId\tNumber of posts.

(ii). Find all posts that are favoured by more than 10 users. You need to output both PostId and the list of UserIds, and each line is in format of:

PostId#UserId1,UserId2,UserId3,…,UserIdn

The lines are sorted according to the NUMERIC values of the PostIds in ascending order. Within each line, the UserIds are sorted according to their NUMERIC values in ascending order.

(Hint: the mkString function is useful to format your output)

You can download the code template at: https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/15011.

You can see the result at:
https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/15014.

Spark GraphX programming

Question 3. Create a graph from a file in GraphX (do this in spark shell).

Download the file tiny-graph.txt from https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/15769. There are many ways of loading a graph from disk files. GraphX provides a function fromEdge () to create a graph from only an RDD of edges, which is the most suitable method for the give format. The steps are as below:

a. Import Graphx relevant classes: import org.apache.spark.graphx._
b. Load the file into an RDD named edges (use sc.textFile())
c. Split the lines in edges and transform each line to a tuple (srcId, dstId, attr), where srcId and dstId are of Long data type, and attr is of Double data type (use map())
d. Transform each tuple to an Edge (use map())
e. Use Graph.fromEdges () function to create a graph
f. Now you can use the created graph to do various tasks. For example, you can check the triplets by: graph.triplets.collect()

The detailed code can be seen at:

https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/16315

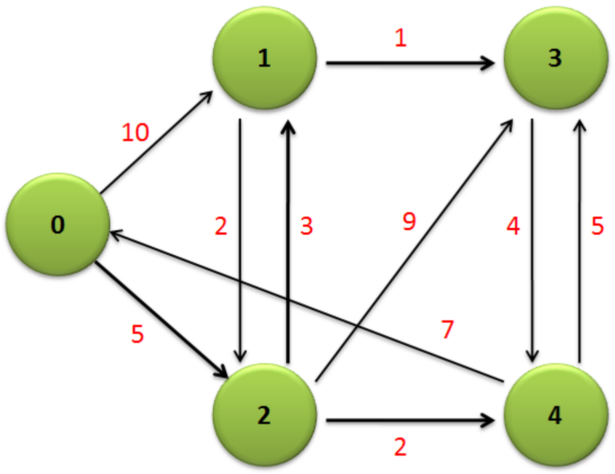
Question 4. Compute the single source shortest distances (do this in Eclipse).

Combine the codes in Question 3 and the codes provided in the lecture slides, and compute the shortest distances with source node 0. Your program should take two parameters: the first is the file location and the second is the source node ID. The code template can be downloaded from:

https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/16315

Question 5. Given a directed graph and a number k, find all vertices that can go back to themselves within k hops. Your program should take a file containing the graph and a value as input. You can use the code template of Question 4 for this question as well, and now the second parameter is the value of k.

For example, give the below example graph and k=2, the output is 1, 2, 3, and 4. 0 is not in the result because it needs 3 hops.



Hint: the idea is that, on each vertex, we use a set to store all the vertices that can reach it. After k iterations, if a vertex is in its own set, we know that this vertex is in the result.

a. Use Set[VertexId] as the vertex attribute
b. Initially, make the sets on all vertices empty
c. When a vertex receives a message (a set of vertices that can reach it), combine the set on the vertex and the message as the new attribute of this vertex
d. When sending out messages on each edge, merge the source vertex and the set on the source as the message to the destination vertex
e. When combining messages from different sources on a vertex, you just need to merge all the messages (sets)
f. To merge two sets *a* and *b*, you can use the ++ operator, and to add one vertex *v* to a set *a*, you can use the + operator
g. Set the maximum number of iterations to k, because we only need to run k rounds to get the results (why?)

Try to solve the problem on your own first. If you have no clue, you can refer to the code at: https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/16316.

It is strongly recommended that you have a try. This question aims to make you have a better understanding of the Pregel operator, and thus to help you solve the second problem of Project 3.