

IERG 4330/ESTR 4316/IEMS 5730 Spring 2022

Homework 5

Release date: Apr 20, 2022

Due date: 11:59:00 pm, May 10, 2022

No late homework will be accepted!

Every Student **MUST** include the following statement, together with his/her signature in the submitted homework.

I declare that the assignment submitted on Elearning system is original except for source material explicitly acknowledged, and that the same or related material has not been previously submitted for another course. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website

<http://www.cuhk.edu.hk/policy/academichonesty/>.

Signed (Student Eddie Christopher Fox III) Date: May 10, 2022

Name Eddie Christopher Fox III SID 1155160788

Submission notice:

- Submit your homework via the elearning system

General homework policies:

A student may discuss the problems with others. However, the work a student turns in must be created COMPLETELY by oneself ALONE. A student may not share ANY written work or pictures, nor may one copy answers from any source other than one's own brain.

Each student **MUST LIST** on the homework paper the **name of every person he/she has discussed or worked with**. If the answer includes content from any other source, the student **MUST STATE THE SOURCE**. Failure to do so is cheating and will result in sanctions. Copying answers from someone else is cheating even if one lists their name(s) on the homework.

If there is information you need to solve a problem but the information is not stated in the problem, try to find the data somewhere. If you cannot find it, state what data you need, make a reasonable estimate of its value and justify any assumptions you make. You will be graded not only on whether your answer is correct but also on whether you have done an intelligent analysis.

IEMS 5730: Spring 2022 Assignment #5

Question 1: GraphFrames

Section 1A: (Final code and submission on page 5)

I refer to the Graph tutorial when writing code for this question [1].

According to [1]'s submission command, we are running GraphFrames 0.7.0 on the IE DIC cluster, but I can only find documentation for version 0.8, so I refer to [2] for GraphFrames documentation. It seems the only change between 0.7 and 0.8 is support for newer Spark and Scala versions, as well as fixing a shortest path error in 0.8.1, but we aren't using shortest path, so we won't be impacted by this.

From reading the Assignment 5 question 1 requirements, it is apparent that the "vertices.tsv" file represents the vertices, while "mooc_actions.tsv" represents the edges. The vertices are already in the correct format, with an id column, however mooc_actions.tsv has the first two columns as USERID and TARGETID instead of src and dst as GraphFrames prefers.

According to [2] we need to use the dataframes API to read the vertices and edges in as dataframes.

```
22/05/07 09:48:03 INFO DataWriteManager: Deleting directory /data1/tmp/spark
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -ls
Found 1 items
drwxr-xr-x  - s1155160788 student          0 2022-05-07 09:48 .sparkStaging
[s1155160788@dicvmd10 Assignment5]$ ls
mooc_actions.tsv  q1a.py  vertices.tsv
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -copyFromLocal mooc_actions.tsv
hdf[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -copyFromLocal vertices.tsv
hdfs d[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -ls
Found 3 items
drwxr-xr-x  - s1155160788 student          0 2022-05-07 09:48 .sparkStaging
-rw-r--r--  3 s1155160788 student    7996988 2022-05-07 10:40 mooc_actions.tsv
-rw-r--r--  3 s1155160788 student     71405 2022-05-07 10:40 vertices.tsv
[s1155160788@dicvmd10 Assignment5]$
```

I copy the mooc_actions.tsv and vertices.tsv files to HDFS so I can reference their path.

I refer to [3] for the command to read csv files into a dataframe, since the tab separated file is like a csv file with a specified tab delimiter. I use the given header and infer the schema. For vertices, this is enough since there is already the id column, but for mooc_actions we will need to rename the columns after reading them in before passing them to GraphFrames.

```

+---+---+
| id|type|
+---+---+
|  0|User|
|  1|User|
|  2|User|
|  3|User|
|  4|User|
|  5|User|
|  6|User|
|  7|User|
|  8|User|
|  9|User|
| 10|User|
| 11|User|
| 12|User|
| 13|User|
| 14|User|
| 15|User|
| 16|User|
| 17|User|
| 18|User|
| 19|User|
+---+---+

```

only showing top 20 rows

```

root
|-- id: integer (nullable = true)
|-- type: string (nullable = true)

```

This confirms the vertices.tsv file is read in correctly.

I refer to [5] for syntax on how to rename the columns of the dataframe, by creating a new dataframe after the initial read, since dataframes are immutable.

Screenshot on final edges on the next page.

```

+---+---+---+
|src|dst|TIMESTAMP|
+---+---+---+
| 0|7047|      0.0|
| 0|7048|      6.0|
| 0|7049|     41.0|
| 0|7048|     49.0|
| 0|7049|     51.0|
| 0|7050|     55.0|
| 0|7051|     59.0|
| 0|7052|     62.0|
| 0|7053|     65.0|
| 0|7054|    113.0|
| 0|7055|    226.0|
| 0|7056|    974.0|
| 0|7047|   1000.0|
| 0|7056|   1172.0|
| 0|7049|   1182.0|
| 0|7048|   1185.0|
| 0|7057|   1687.0|
| 1|7057|   7262.0|
| 1|7048|   7266.0|
| 1|7049|   7273.0|
+---+---+---+
only showing top 20 rows

```

```

root
|-- src: integer (nullable = true)
|-- dst: integer (nullable = true)
|-- TIMESTAMP: double (nullable = true)

```

This confirms that the mooc_actions.tsv file is being read correctly and that the columns have the correct name.

For the first four subsections of 1A, we simply invoke the `.count()` function at the right spot, but for in-degree and out-degree, we either need to do grouping or ordering / sorting.

I at first tried using `.groupBy()` and `.max()` after calling `inDegrees` and `outDegrees`, as in the documentation [2] however this only returned the highest number of in-degree / out-degree, and not the corresponding vertex ID. Using [6], I was able to accomplish the desired function by using sort in descending order, and then limiting the shown rows to 1.

Final code and submission on next page.

Final Question 1A Code:

```

q1a.py 9+ X
C: > Users > Eddie > Documents > School > CUHK > 2021 - 2022 Term 2 > IEMS 5730 - Big Data Systems and Information Processing > Week 16 > Assignment 5 > q1a.py > ...
1 from pyspark.sql import SQLContext
2 from pyspark import SparkConf, SparkContext
3 from pyspark.sql.session import SparkSession
4 from pyspark.sql.functions import *
5 from graphframes import *
6
7 sc = SparkContext.getOrCreate()
8 sqlContext = SQLContext(sc)
9 spark = SparkSession(sc)
10
11 # Read files to create vertex and edge dataframes
12 vertices = spark.read.options(header='True', inferSchema='True', delimiter='\t').csv("hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/vertices.tsv")
13 edges_initial = spark.read.options(header='True', inferSchema='True', delimiter='\t').csv("hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/mooc_actions.tsv")
14
15 # Rename columns on edge dataframe to be compliant with GraphFrame requirements
16 edges = edges_initial.withColumnRenamed("USERID","src").withColumnRenamed("TARGETID","dst")
17
18 # Create the GraphFrame
19 graph = GraphFrame(vertices, edges)
20
21 # Question 1A Queries
22 print "Number of vertices: %d" % graph.vertices.count()
23 print "Number of users: %d" % graph.vertices.filter("type = 'User'").count()
24 print "Number of course activities: %d" % graph.vertices.filter("type = 'Course Activity'").count()
25 print "Number of edges: %d" % graph.edges.count()
26 graph.inDegrees.sort("inDegree", ascending=False).show(1)
27 graph.outDegrees.sort("outDegree", ascending=False).show(1)

```

Submission command derived from [1].

“spark-submit --repositories https://repos.spark-packages.org --packages graphframes:graphframes:0.7.0-spark2.3-s_2.11 --master yarn --deploy-mode cluster q1a.py”

Question 1A Final Submission:

```

Number of vertices: 7144
Number of users: 7047
Number of course activities: 97
Number of edges: 411749
+---+-----+
| id|inDegree|
+---+-----+
|7055|   19474|
+---+-----+
only showing top 1 row

+---+-----+
| id|outDegree|
+---+-----+
|1181|     505|
+---+-----+
only showing top 1 row

```

The last two items show the id corresponding to the vertex with the highest in-degree and out-degree, along with what that in-degree or out-degree was.

I validated the first four results to confirm accuracy by referring to the raw vertex and edge lists. The results also looked reasonable for out-degree. I could see a user taking 505 courses / course activities if they were an enthusiast. But I didn't understand how a course activity could have 19,474 users when there was only 7047 users. While searching through the raw edge list, I

discovered that users were able to take the same course / have the same course activity more than once.

682	96	7048	69487.0
683	97	7050	69492.0
684	90	7057	69506.0
685	97	7051	69508.0
686	97	7060	69508.0
687	98	7048	69508.0
688	54	7053	69519.0
689	54	7063	69519.0
690	91	7050	69521.0
691	96	7057	69533.0
692	59	7064	69534.0
693	59	7065	69534.0
694	90	7057	69535.0
695	96	7048	69541.0

In this screenshot, we see user 96 engaging in course activity 7048 twice in quick succession, so it is possible for the indegrees to exceed the number of users, so the final results should be valid.

Section 1B:

The first 20 lines are identical to Section 1A. Here, I refer to the GraphFrame documentation's subgraph section [2] and use filterEdges twice to adhere to the timestamp conditions, then use dropIsolatedVertices() as requested in the Assignment 5 PDF. Finally, we apply the same operations we used in section 1A to count the vertices and edges of this newly generated subgraph.

Submission command:

```
"spark-submit --repositories https://repos.spark-packages.org --packages
graphframes:graphframes:0.7.0-spark2.3-s_2.11 --master yarn --deploy-mode cluster q1b.py"
```

Final code and submission on next page.

Section 1B Final Code and Final Submission:

```

q1b.py 6 X
C:\Users\Eddie> Documents > School > CUHK > 2021 - 2022 Term 2 > IEMS 5730 - Big Data Systems and Information Processing > Week 16 > Assignment 5 > q1b.py > ...
1  from pyspark.sql import SQLContext
2  from pyspark import SparkConf, SparkContext
3  from pyspark.sql.session import SparkSession
4  from pyspark.sql.functions import *
5  from graphframes import *
6
7  sc = SparkContext.getOrCreate()
8  sqlContext = SQLContext(sc)
9  spark = SparkSession(sc)
10
11 # Read files to create vertex and edge dataframes
12 vertices = spark.read.options(header='True', inferSchema='True', delimiter='\t').csv("hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/vertices.tsv")
13 edges_initial = spark.read.options(header='True', inferSchema='True', delimiter='\t').csv("hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/mooc_actions.tsv")
14
15 # Rename columns on edge dataframe to be compliant with GraphFrame requirements
16 edges = edges_initial.withColumnRenamed("USERID","src").withColumnRenamed("TARGETID","dst")
17
18 # Create the GraphFrame
19 graph = GraphFrame(vertices, edges)
20
21 # Question 1B Queries
22 subGraph = graph.filterEdges("TIMESTAMP >= 10000").filterEdges("TIMESTAMP <= 50000").dropIsolatedVertices()
23 print "Number of nodes in subgraph: %d" % subGraph.vertices.count()
24 print "Number of edges in subgraph: %d" % subGraph.edges.count()

```

```

Number of nodes in subgraph: 49
Number of edges in subgraph: 243

```

I validate the number of edges in the subgraph by looking at the edge list and noting the line numbers where edges satisfying the timestamp conditions begin and end. This is Line 27 – 269, which is $(269 - 27 + 1) = 243$ edges, which matches. It is not as easy to validate the nodes but it should be accurate if the edges are.

Section 1C (Final code and submission on page 10-11):

I first verbally break down each pattern to determine what needs to be done. Afterwards, I will show a screenshot of the calculated motif count, then a visualization of the output through the first 20 calculated patterns. At the end of all the motifs is the final code and submission. When writing the code, I mainly refer to [2]. This section is more about thinking and understanding than coding, so I did not need many references.

Submission command: “spark-submit --repositories https://repos.spark-packages.org --packages graphframes:graphframes:0.7.0-spark2.3-s_2.11 --master yarn --deploy-mode cluster q1c.py”

Motif I:

Node A points to Node B through Edge 1

Node C points to Node B through Edge 2

ID of node A and C ID have to be distinct

Timestamp of edge 1 needs to be less or equal to timestamp of edge 2.

```

Log Length: 66
Number of times motif I occurred: 2372

```

	a	e1	b	c	e2
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[33, User]	[33, 7048, 49692.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[23, User]	[23, 7048, 49040.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[32, User]	[32, 7048, 47359.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[31, User]	[31, 7048, 45708.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[30, User]	[30, 7048, 45092.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[29, User]	[29, 7048, 43662.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[29, User]	[29, 7048, 43615.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[28, User]	[28, 7048, 42489.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[27, User]	[27, 7048, 41947.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[27, User]	[27, 7048, 41934.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[26, User]	[26, 7048, 41882.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[16, User]	[16, 7048, 41604.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[25, User]	[25, 7048, 41049.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[24, User]	[24, 7048, 40670.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[23, User]	[23, 7048, 40325.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[21, User]	[21, 7048, 39627.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[20, User]	[20, 7048, 39612.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[20, User]	[20, 7048, 39606.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[14, User]	[14, 7048, 39447.0]	
[2, User]	[2, 7048, 37868.0]	[7048, Course Act...	[14, User]	[14, 7048, 39438.0]	

^ Visualization of Motif I output

Verbal example: User 2 takes course activity 7048, then another user takes the same activity later in time.

Motif II:

Node A points to Node B through Edge 1

Node B points to Node C through Edge 2

ID of node A and C have to be distinct

Timestamp order: Edge 1 <= Edge 2

Number of times motif II occurred: 0

	a	e1	b	e2	c

^ Visualization of Motif II output

Verbal example: Hypothetically, user A takes course activity B, then the same course activity takes a user??

There is 0 instances of this occurring because there is never an example of an activity having an edge that points at a user. Only users can point at activities and activities never point at anything.

Motif III:

Node A points to Node B through Edge 1

Node C points to Node D through Edge 2

Node A points to Node D through Edge 3

Node C points to Node B through Edge 4

ID of node A and C have to be distinct

ID of node B and D have to be distinct

Timestamp order: Edge 1 <= Edge 2 <= Edge 3 <= Edge 4

Number of times motif III occurred: 215

a	e1	b	c	e2	d	e3	e4
[6, User]	[6, 7048, 38218.0]	[7048, Course Act...	[14, User]	[14, 7060, 39133.0]	[7060, Course Act...	[6, 7060, 39445.0]	[14, 7048, 39447.0]
[6, User]	[6, 7050, 38261.0]	[7050, Course Act...	[7, User]	[7, 7048, 38287.0]	[7048, Course Act...	[6, 7048, 38308.0]	[7, 7050, 38583.0]
[6, User]	[6, 7048, 38308.0]	[7048, Course Act...	[14, User]	[14, 7060, 39133.0]	[7060, Course Act...	[6, 7060, 39445.0]	[14, 7048, 39447.0]
[6, User]	[6, 7049, 38327.0]	[7049, Course Act...	[14, User]	[14, 7060, 39133.0]	[7060, Course Act...	[6, 7060, 39445.0]	[14, 7049, 39513.0]
[6, User]	[6, 7054, 38340.0]	[7054, Course Act...	[13, User]	[13, 7060, 39105.0]	[7060, Course Act...	[6, 7060, 39445.0]	[13, 7054, 39591.0]
[6, User]	[6, 7054, 38606.0]	[7054, Course Act...	[13, User]	[13, 7060, 39105.0]	[7060, Course Act...	[6, 7060, 39445.0]	[13, 7054, 39591.0]
[6, User]	[6, 7056, 38623.0]	[7056, Course Act...	[13, User]	[13, 7060, 39105.0]	[7060, Course Act...	[6, 7060, 39445.0]	[13, 7056, 39667.0]
[6, User]	[6, 7056, 38623.0]	[7056, Course Act...	[13, User]	[13, 7060, 39105.0]	[7060, Course Act...	[6, 7060, 39445.0]	[13, 7056, 39623.0]
[14, User]	[14, 7049, 39065.0]	[7049, Course Act...	[16, User]	[16, 7048, 39193.0]	[7048, Course Act...	[14, 7048, 39447.0]	[16, 7049, 41611.0]
[14, User]	[14, 7049, 39065.0]	[7049, Course Act...	[16, User]	[16, 7048, 39193.0]	[7048, Course Act...	[14, 7048, 39438.0]	[16, 7049, 41611.0]
[14, User]	[14, 7049, 39065.0]	[7049, Course Act...	[16, User]	[16, 7048, 39193.0]	[7048, Course Act...	[14, 7048, 39370.0]	[16, 7049, 41611.0]
[13, User]	[13, 7051, 39105.0]	[7051, Course Act...	[14, User]	[14, 7056, 39157.0]	[7056, Course Act...	[13, 7056, 39623.0]	[14, 7051, 39627.0]
[13, User]	[13, 7051, 39105.0]	[7051, Course Act...	[14, User]	[14, 7056, 39157.0]	[7056, Course Act...	[13, 7056, 39623.0]	[14, 7051, 39624.0]
[13, User]	[13, 7051, 39105.0]	[7051, Course Act...	[14, User]	[14, 7056, 39133.0]	[7056, Course Act...	[13, 7056, 39623.0]	[14, 7051, 39627.0]
[13, User]	[13, 7051, 39105.0]	[7051, Course Act...	[14, User]	[14, 7056, 39133.0]	[7056, Course Act...	[13, 7056, 39623.0]	[14, 7051, 39624.0]
[13, User]	[13, 7051, 39105.0]	[7051, Course Act...	[14, User]	[14, 7054, 39162.0]	[7054, Course Act...	[13, 7054, 39591.0]	[14, 7051, 39627.0]
[13, User]	[13, 7051, 39105.0]	[7051, Course Act...	[14, User]	[14, 7054, 39162.0]	[7054, Course Act...	[13, 7054, 39591.0]	[14, 7051, 39624.0]
[14, User]	[14, 7050, 39113.0]	[7050, Course Act...	[17, User]	[17, 7057, 39258.0]	[7057, Course Act...	[14, 7057, 39543.0]	[17, 7050, 41319.0]
[14, User]	[14, 7050, 39113.0]	[7050, Course Act...	[17, User]	[17, 7048, 39220.0]	[7048, Course Act...	[14, 7048, 39447.0]	[17, 7050, 41319.0]
[14, User]	[14, 7050, 39113.0]	[7050, Course Act...	[17, User]	[17, 7048, 39220.0]	[7048, Course Act...	[14, 7048, 39438.0]	[17, 7050, 41319.0]

only showing top 20 rows

^ Visualization of Motif III output

Verbal example: User 6 takes course activity 7048. User 14 takes course activity 7060. Then user 6 takes course activity 7060. Finally User 14 takes course activity 7048.

Motif IV:

Node A points to Node B through Edge 1

Node C points to Node D through Edge 2

Node A points to Node E through Edge 3

Node C points to Node E through Edge 4

ID of Node A and Node C have to be distinct

ID of Node B, D, and E have to be distinct. (To write this one, we need to break it up into 3 filters. B can't equal D, B can't equal E, D can't equal E).

Timestamp order: Edge 1 <= Edge 2 <= Edge 3 <= Edge 4

Number of times motif IV occurred: 5076

	a	e1	b	c	e2	d	e3	e	e4
[3, User]	[3, 7048, 37953.0]	[7048, Course Act...	[4, User]	[4, 7050, 38018.0]	[7050, Course Act...	[3, 7060, 38246.0]	[7060, Course Act...	[4, 7060, 38725.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7050, 39320.0]	[7050, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[17, 7063, 41979.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7057, 39258.0]	[7057, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[17, 7063, 41979.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7050, 39320.0]	[7050, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[17, 7063, 41646.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7057, 39258.0]	[7057, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[17, 7063, 41646.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7050, 39320.0]	[7050, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[17, 7063, 41310.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7057, 39258.0]	[7057, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[17, 7063, 41310.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[18, User]	[18, 7050, 39273.0]	[7050, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[18, 7063, 41097.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[20, User]	[20, 7050, 39624.0]	[7050, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[20, 7063, 40790.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[20, User]	[20, 7049, 39610.0]	[7049, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[20, 7063, 40790.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[20, User]	[20, 7050, 39624.0]	[7050, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[20, 7063, 40732.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[20, User]	[20, 7049, 39610.0]	[7049, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[20, 7063, 40732.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[18, User]	[18, 7050, 39273.0]	[7050, Course Act...	[4, 7063, 39882.0]	[7063, Course Act...	[18, 7063, 40687.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7050, 39320.0]	[7050, Course Act...	[4, 7063, 39630.0]	[7063, Course Act...	[17, 7063, 41979.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7057, 39258.0]	[7057, Course Act...	[4, 7063, 39630.0]	[7063, Course Act...	[17, 7063, 41979.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7050, 39320.0]	[7050, Course Act...	[4, 7063, 39630.0]	[7063, Course Act...	[17, 7063, 41646.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7057, 39258.0]	[7057, Course Act...	[4, 7063, 39630.0]	[7063, Course Act...	[17, 7063, 41646.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7050, 39320.0]	[7050, Course Act...	[4, 7063, 39630.0]	[7063, Course Act...	[17, 7063, 41310.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[17, User]	[17, 7057, 39258.0]	[7057, Course Act...	[4, 7063, 39630.0]	[7063, Course Act...	[17, 7063, 41310.0]	
[4, User]	[4, 7048, 37969.0]	[7048, Course Act...	[18, User]	[18, 7050, 39273.0]	[7050, Course Act...	[4, 7063, 39630.0]	[7063, Course Act...	[18, 7063, 41097.0]	

^ Visualization of Motif IV output

User 3 takes course activity 7048. User 4 takes course activity 7050. User 3 takes course activity 7060. User 4 takes course activity 7060.

Section 1C Final Code:

```

qlc.py 9+ X
C: > Users > Eddie > Documents > School > CUHK > 2021 - 2022 Term 2 > IEMS 5730 - Big Data Systems and Information Processing > Week 16 > Assignment 5 > qlc.py ...

1 from pyspark.sql import SQLContext
2 from pyspark import SparkConf, SparkContext
3 from pyspark.sql.session import SparkSession
4 from pyspark.sql.functions import *
5 from graphframes import *
6
7 sc = SparkContext.getOrCreate()
8 sqlContext = SQLContext(sc)
9 spark = SparkSession(sc)
10
11 # Read files to create vertex and edge dataframes
12 vertices = spark.read.options(header='True', inferSchema='True', delimiter='\\t').csv("hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/vertices.tsv")
13 edges_initial = spark.read.options(header='True', inferSchema='True', delimiter='\\t').csv("hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/mooc_actions.tsv")
14
15 # Rename columns on edge dataframe to be compliant with GraphFrame requirements
16 edges = edges_initial.withColumnRenamed("USERID","src").withColumnRenamed("TARGETID","dst")
17
18 # Create the GraphFrame
19 graph = GraphFrame(vertices, edges)
20
21 # Create the subgraph from Question 1B
22 subgraph = graph.filterEdges("TIMESTAMP >= 10000").filterEdges("TIMESTAMP <= 50000").dropIsolatedVertices()
23
24 # Question 1C Queries
25 motif_1 = subgraph.find("(a)-[e1]->(b); (c)-[e2]->(b)")
26 filter_1 = motif_1.filter("a.id != c.id").filter("e1.TIMESTAMP <= e2.TIMESTAMP")
27 print "Number of times motif I occurred: %d" % filter_1.count()
28
29 motif_2 = subgraph.find("(a)-[e1]->(b); (b)-[e2]->(c)")
30 filter_2 = motif_2.filter("a.id != c.id").filter("e1.TIMESTAMP <= e2.TIMESTAMP")
31 print "Number of times motif II occurred: %d" % filter_2.count()
32
33 motif_3 = subgraph.find("(a)-[e1]->(b); (c)-[e2]->(d); (a)-[e3]->(d); (c)-[e4]->(b)")
34 filter_3 = motif_3.filter("a.id != c.id").filter("b.id != d.id")\
35 .filter("e1.TIMESTAMP <= e2.TIMESTAMP").filter("e2.TIMESTAMP <= e3.TIMESTAMP").filter("e3.TIMESTAMP <= e4.TIMESTAMP")
36 print "Number of times motif III occurred: %d" % filter_3.count()

```

```

24 # Question 1C Queries
25 motif_1 = subGraph.find("(a)-[e1]->(b); (c)-[e2]->(b)")
26 filter_1 = motif_1.filter("a.id != c.id").filter("e1.TIMESTAMP <= e2.TIMESTAMP")
27 print "Number of times motif I occurred: %d" % filter_1.count()
28
29 motif_2 = subGraph.find("(a)-[e1]->(b); (b)-[e2]->(c)")
30 filter_2 = motif_2.filter("a.id != c.id").filter("e1.TIMESTAMP <= e2.TIMESTAMP")
31 print "Number of times motif II occurred: %d" % filter_2.count()
32
33 motif_3 = subGraph.find("(a)-[e1]->(b); (c)-[e2]->(d); (a)-[e3]->(d); (c)-[e4]->(b)")
34 filter_3 = motif_3.filter("a.id != c.id").filter("b.id != d.id")\
35 .filter("e1.TIMESTAMP <= e2.TIMESTAMP").filter("e2.TIMESTAMP <= e3.TIMESTAMP").filter("e3.TIMESTAMP <= e4.TIMESTAMP")
36 print "Number of times motif III occurred: %d" % filter_3.count()
37
38 motif_4 = subGraph.find("(a)-[e1]->(b); (c)-[e2]->(d); (a)-[e3]->(e); (c)-[e4]->(e)")
39 filter_4 = motif_4.filter("a.id != c.id").filter("b.id != d.id").filter("b.id != e.id").filter("d.id != e.id")\
40 .filter("e1.TIMESTAMP <= e2.TIMESTAMP").filter("e2.TIMESTAMP <= e3.TIMESTAMP").filter("e3.TIMESTAMP <= e4.TIMESTAMP")
41 print "Number of times motif IV occurred: %d" % filter_4.count()

```

Section 1C Final Submission:

Log Length: 100

```

Number of times motif I occurred: 2372
Number of times motif II occurred: 0
Number of times motif III occurred: 215
Number of times motif IV occurred: 5076

```

Question 2: GraphX

Section 2A:

```

build.sbt x
1 ThisBuild / version := "0.1"
2
3 ThisBuild / scalaVersion := "2.11.8"
4
5 scalaVersion := "2.11.8"
6 val sparkVersion = "2.3.0"
7
8 libraryDependencies ++= Seq(
9   "org.apache.spark" %% "spark-core" % sparkVersion,
10  "org.apache.spark" %% "spark-sql" % sparkVersion,
11  "org.apache.spark" %% "spark-graphx" % sparkVersion
12 )
13 lazy val root = (project in file("."))
14   .settings(
15     name := "Assignment5"
16   )

```

The build.sbt file is above. I take the one from Assignment 4 and adjust the dependencies to include spark-graphx as suggested in the tutorial [1], while removing the kafka ones. The rest of the process, such as creating the object classes and using sbt package to compile the jar file, then uploading the jar file to the IE DIC cluster, then to HDFS is the same as before, so I will not go over that again unless notably different.

While programming GraphX, I read the Spark 2.3.0 version of the GraphX programming guide [7].

```
[s1155160788@dicvmd10 Assignment5]$ ls
dag_edge_list.txt  edge_list.txt  mooc_actions.tsv  q1a.py  q1b.py  q1c.py  vertices.tsv
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -copyFromLocal edge_list.txt
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -copyFromLocal dag_edge_list.txt
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -ls
Found 5 items
drwxr-xr-x  - s1155160788 student      0 2022-05-07 21:41 .sparkStaging
-rw-r--r--  3 s1155160788 student    2142233 2022-05-08 15:53 dag_edge_list.txt
-rw-r--r--  3 s1155160788 student    14773350 2022-05-08 15:53 edge_list.txt
-rw-r--r--  3 s1155160788 student     7996988 2022-05-07 10:40 mooc_actions.tsv
-rw-r--r--  3 s1155160788 student      71405 2022-05-07 10:40 vertices.tsv
[s1155160788@dicvmd10 Assignment5]$
```

I first copy the text files for this question into HDFS.

Submission command:

```
“spark-submit \
    --class Question2.A \
    --master yarn \
    --deploy-mode cluster \
    hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/assignment5_2.11-0.1.jar”
```

I primarily reference the GraphX documentation [7] to write this code, especially for finding the vertices with the highest in degree and out degree, where there is directly code for this in the documentation. For printing operations, I seek advice from [8] for formatting. The definition code from max is directly taken from [7].

Final code for Section 2A on next page.

Final Code for Section 2A:

```

1 package Question2
2 import org.apache.spark._
3 import org.apache.spark.graphx._
4 import org.apache.spark.rdd.RDD
5 import org.apache.spark.SparkContext
6 import org.apache.spark.graphx.GraphLoader
7
8 object A {
9   def main(args: Array[String]) {
10
11     def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
12       if (a._2 > b._2) a else b
13     }
14
15     // Create Spark Context and load file
16     val sc = new SparkContext()
17     val graph = GraphLoader.edgeListFile(sc, path = "hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/edge_list.txt")
18
19     // Question A queries
20     val vertices_count = graph.vertices.count()
21     val edges_count = graph.edges.count()
22     val maxInDegree = graph.inDegrees.reduce(max)
23     val maxOutDegree = graph.outDegrees.reduce(max)
24
25     println(s"Number of vertices: $vertices_count")
26     println(s"Number of edges: $edges_count")
27     println(s"Vertex with the largest in-degree: ${maxInDegree._1}, with an in-degree of ${maxInDegree._2}")
28     println(s"Vertex with the largest out-degree: ${maxOutDegree._1}, with an out-degree of ${maxOutDegree._2}")
29
30     sc.stop()
31   }
}

```

Final Submission for Section 2A:

```

Number of vertices: 169343
Number of edges: 1166243
Vertex with the largest in-degree: 1353, with an in-degree of 13155
Vertex with the largest out-degree: 72253, with an out-degree of 436

```

I validate the results of the first two through ctrl+f operations in the edge_list.txt file. The number of edges corresponds to the number of lines in the file, and I can find references to edge ID 169342 (starts at ID 0, so 169343 edges), but there is nothing for ID 169343, indicating the number of edges is correct. I cannot directly verify the in and out degree answers, but the results look reasonable. Ctrl+f'ing the edge list for the vertex ID will result in slightly more results than the calculated in-degree and out-degree because the in-degree one will be pointing to some edges and the out-degree one will have some edges pointing to it.

Section 2B:

For Part B, I reuse the initial code from A but take out the definition of max(). During the GraphLoader call of edgeListFile(), I set canonical orientation to true so that it is structured such

that $\text{srcID} < \text{dstID}$, as it is noted in [7] that canonical orientation is required for connected components algorithms.

Submission command:

```
“spark-submit \
    --class Question2.B \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 10G \
    hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/assignment5_2.11-0.1.jar”
```

For subsection I with connected components, I use code from [9] to find the number of connected components in a graph. Connected components will label every vertex with the id of the lowest numbered vertex [7], so by counting the number of distinct labels, we get the number of connected components.

To find the number of vertices belonging to the largest connected component, I use code from [10], which advocates using `countByKey` in combination with `map`. What this does is add 1 for each vertex that has the same label (with the lowest id vertex of the connected component), which effectively gives us the number of vertices in each connected component. We then print the results of `countByKey`. The tuples are in the format (id, vertex count) where id is the lowest number vertex id in the connected component, and vertex count is the number of vertices in the connected component.

For strongly connected components, we use similar code. I refer to [11] as suggested by the TA, as it has implementation details. I then use the code from [12], which is similar to the previous connected component counting code to find the number of distinct vertex id's, which gives us the number of strongly connected components. I find that the results differ based on if canonical orientation is enabled or not, and also depending on the number of iterations. I detail these later.

I set driver memory explicitly to 10GB because I get resource errors when using the default amount of allocated memory.

Section 2B final code on next page.

Section 2B Final Code:

```

1 package Question2
2 import org.apache.spark._
3 import org.apache.spark.graphx._
4 import org.apache.spark.rdd.RDD
5 import org.apache.spark.SparkContext
6 import org.apache.spark.graphx.GraphLoader
7
8 object B {
9   def main(args: Array[String]) {
10
11     // Create Spark Context and load file
12     val sc = new SparkContext()
13     val graph = GraphLoader.edgeListFile(sc, path = "hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/edge_list.txt")
14
15     // Question B queries
16     val connectedGraph = graph.connectedComponents()
17     val connected_count = connectedGraph.vertices.map{ case (_, cc) => cc }.distinct.count()
18     println("Subsection I: Connected Components")
19     println(s"Number of connected components in the citation network: $connected_count")
20     println("The connected components are below.")
21     println("Format: (Lowest vertex ID in connected component, Number of vertices in connected component)")
22     connectedGraph.vertices.map(x => (x._2, x._2)).countByKey().foreach(println)
23
24     println("Subsection II: Strongly Connected Components")
25     val stronglyConnectedGraph = graph.stronglyConnectedComponents(numiter = 5)
26     val stronglyConnected_count = stronglyConnectedGraph.vertices.map(x => x._2).distinct.count
27     println(s"Number of strongly connected components in the citation network: $stronglyConnected_count")
28
29     sc.stop()
30   }
31 }

```

Section 2B Final Submission:

```

Subsection I: Connected Components
Number of connected components in the citation network: 1
The connected components are below.
Format: (Lowest vertex ID in connected component, Number of vertices in connected component)
(0,169343)
Subsection II: Strongly Connected Components
Number of strongly connected components in the citation network: 141346

```

For connected components, the result is always that there is a single connected component in the citation network. This means that every page can reach every other page through some chain of citations. This makes sense as being possible, because academic papers tend to cite many sources. The only way a paper could not be connected to the rest is if it cites no papers and no papers cite it. Although an unpopular or new paper might not be cited, they would surely cite the previous works, connecting them to every other node by creating a link that points to other papers.

The number of vertices in the largest (and only) connected component is 169,343, which makes sense because that is how many vertices there are in the graph to begin with, and all the vertices are connected into one big connected component.

For strongly connected components, the result depends on if canonical orientation is disabled or enabled, and also depends on the number of iterations chosen. I detail my findings in the table below.

# of iterations	Canonical orientation	# of strongly connected components	Runtime
1	Enabled	169,343	2.5 minutes
1	Disabled	169,343	19 seconds
3	Enabled	169,343	2.5 minutes
3	Disabled	142,264	3.5 minutes
5	Enabled	169,343	2.5 minutes
5	Disabled	141,346	43 minutes

We can observe several things from these results.

Firstly, when canonical orientation is enabled, it always takes 2.5 minutes, and always outputs a result of 169,343. This is also the same result as when canonical orientation is disabled with 1 iteration.

When canonical orientation is disabled, the amount of strongly connected components decreases but the runtime seems to increase exponentially.

I believe that the count of strongly connected components may continue to decrease as the number of iterations decrease, but there is diminishing returns as it gets closer to converging on the true value. The count dropped 27,079 between 1 and 3 iterations, but only dropped 918 between 3 iterations and 5, while taking over 17 times longer to run.

We can probably say that the true value of the number of strongly connected components is **around 141,000**, with more iterations increasingly approaching the true value. This is a sensible value. We would expect the value to be close to the total number of vertices because there is a stringent requirement for strongly connected components in that every paper has to cite every other paper within the strongly connected component. The reason it isn't exactly equal to the number of vertices is because there would be small pockets of maybe 2-50 papers at a time where all of them cite each other. If 30 papers all cited each other, then we would subtract 29 from the value of strongly connected components because they are all under 1.

Section 2C:

I refer to [13] for the syntax for personalized pagerank, as it is defined under the graphops class. [14] reminds me of how to sort by the second column of an RDD in descending order. I use the documentation [7] for inspiration + some code for the join operations and the subgraph operations.

Submission command:

```
“spark-submit \
    --class Question2.C \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 10G \
    hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/assignment5_2.11-0.1.jar”
```

Final Section 2C Code:

```
1 package Question2
2 import org.apache.spark._
3 import org.apache.spark.graphx._
4 import org.apache.spark.rdd.RDD
5 import org.apache.spark.SparkContext
6 import org.apache.spark.graphx.GraphLoader
7
8 object C {
9   def main(args: Array[String]) {
10
11     // Create Spark Context and load file
12     val sc = new SparkContext()
13     val graph = GraphLoader.edgeListFile(sc, path = "hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/edge_list.txt")
14
15     // Question C queries
16     val pageRank4330 = graph.personalizedPageRank( src = 4330, tol = 0.000000001, resetProb = 0.15)
17     val pageRank5730 = graph.personalizedPageRank( src = 5730, tol = 0.000000001, resetProb = 0.15)
18
19     println("Subsection I:")
20     println("Top 20 Pagerank vertices for Vertex 4330:")
21     pageRank4330.vertices.sortBy(_._2, ascending = false).take( num = 20).foreach(println)
22
23     println("\nTop 20 Pagerank vertices for Vertex 5730:")
24     pageRank5730.vertices.sortBy(_._2, ascending = false).take( num = 20).foreach(println)
25
26     println("Subsection II:")
27     val top2000Array = pageRank5730.vertices.sortBy(_._2, ascending = false).take( num = 2000)
28     val top2000RDD = sc.parallelize(top2000Array)
29     val newGraph = graph.outerJoinVertices(top2000RDD)((vid, _, rank) => rank.getOrElse(-1))
30     val subgraph = newGraph.subgraph(vpred = (id, attr) => attr != -1)
31     val subgraph_vertex_count = subgraph.vertices.count()
32     val subgraph_edge_count = subgraph.edges.count()
33     println(s"The subgraph has $subgraph_vertex_count vertices and $subgraph_edge_count edges")
34     sc.stop()
35   }
36 }
```

I tried different values of tolerance for the personalizedPageRank, but that did not change the results much. I was shocked to see that node 5730 only had 4 nodes with a PageRank above 0, especially since in Subsection II we are asked to find a subgraph of the top 2000 pagerank nodes

for 5730, but I confirmed through the documentation that I coded the pagerank correctly, so this must be intentional.

For the subgraph question, my initial plan was to use `joinVertices()` with the original graph to attach the pagerank property to vertices with id's corresponding to the top 2000, then create a subgraph using a vertex predicate that checked for the existence of the pagerank attribute, so that only the same nodes in the top 2000 would be part of the subgraph. I struggled because after taking the top 2000 results with `take(n)`, it became an array of (Vertex ID, Double) rather than an RDD, and `joinVertices()` only takes RDD's.

Learning from [15], I recalled that I could run `parallelize` on the array to transform it into an RDD that `joinVertices()` would take. I ultimately ended up using `outerJoinVertices()` to create a new graph joining the original graph with the top 2000 in RDD form, but I used `getOrElse` so that any vertex without the pagerank property would have their rank set to -1, which would be an impossible pagerank in normal operation. The join code is copied from the GraphX documentation [7] but with the argument for `getOrElse` being -1 instead of 0. This simplified the subgraph predicate checking because rather than needing to figure out how to check for the existence of the attribute, I just needed to ensure it was not equal to -1.

Final submission for Section 2C on next page.

Final Submission for Section 2C:**Subsection I:**

Top 20 Pagerank vertices for Vertex 4330:

(4330,0.2246951415508737)
 (153677,0.06426497273659798)
 (16921,0.04415768456202822)
 (123933,0.04233866462308503)
 (162330,0.038909865029924204)
 (63804,0.030343709693851462)
 (19645,0.028883541440823536)
 (113389,0.028400958513268838)
 (26614,0.027484307892014042)
 (133211,0.02728441004546323)
 (111306,0.02728441004546323)
 (71148,0.019018937097514576)
 (69228,0.016663636608962234)
 (81746,0.015419023760446283)
 (112464,0.015316167346685061)
 (62510,0.01418834397727014)
 (42645,0.013904279041980445)
 (119218,0.013811360099136137)
 (96770,0.012529064022003327)
 (77078,0.012047940049048949)

Top 20 Pagerank vertices for Vertex 5730:

(5730,0.45223289994347093)
 (102862,0.19219898247597514)
 (165911,0.19219898247597514)
 (141857,0.16336913510457887)
 (65722,0.0)
 (108150,0.0)
 (139526,0.0)
 (91902,0.0)
 (154038,0.0)
 (68522,0.0)
 (23776,0.0)
 (129434,0.0)
 (153030,0.0)
 (32676,0.0)
 (53926,0.0)
 (103184,0.0)
 (4926,0.0)
 (38926,0.0)
 (51620,0.0)
 (63852,0.0)

Subsection II:

The subgraph has 2000 vertices and 167 edges

The format for Subsection I is (Node ID, Pagerank)

The stdout logs confirm the subgraph has exactly 2000 nodes, and the number of edges is reasonable at 167 edges. Because only 4 nodes actually had a pagerank above 0, we would expect the remaining 1996 to be arbitrarily decided and thus, probably spread about the graph and lacking many edges to other nodes.

Section 2D:

Submission command:

```
“spark-submit \
    --class Question2.D \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 10G \
    hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/assignment5_2.11-0.1.jar”
```

I refer to [16] as the documentation for LabelPropagation and all its methods, including run. Calling it is fairly straightforward, just using the graph and 50 as the number of max steps. The assignment says to find the number of vertices in the largest “communities”, plural instead of singular, so I find the top 10. I accomplish this by taking the community graph generated by LabelPropagation and map the vertices in the format (CommunityID, 1), then perform a word count at the Spark RDD level using reduceByKey. I initially tried countByKey, but was unable to sort it because countByKey is an action while reduceByKey is a transformation, as I learned in [17]. Afterwards, I was able to sort by the accumulated vertex count in descending order and then print the top 10 results.

Final Section 2D code:

```
1 package Question2
2 import org.apache.spark._
3 import org.apache.spark.graphx._
4 import org.apache.spark.rdd.RDD
5 import org.apache.spark.SparkContext
6 import org.apache.spark.graphx.GraphLoader
7 import org.apache.spark.graphx.lib.LabelPropagation
8
9 object D {
10 def main(args: Array[String]) {
11
12     // Create Spark Context and load file
13     val sc = new SparkContext()
14     val graph = GraphLoader.edgeListFile(sc, path = "hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/edge_list.txt")
15
16     // Question D queries
17     val communityGraph = LabelPropagation.run(graph, maxSteps = 50)
18     val community_count = communityGraph.vertices.map{ case (_, community) => community }.distinct.count()
19     println(s"Number of communities: $community_count")
20     println("Top 10 Largest communities: Format: (Community ID, Number of Vertices)")
21     communityGraph.vertices.map(x => (x._2, 1)).reduceByKey(_ + _).sortBy(_._2, ascending = false).take(num = 10).foreach(println)
22     sc.stop()
23 }
24 }
```

Final Section 2D Submission:

Number of communities: 14107

Top 10 Largest communities: Format: (Community ID, Number of Vertices)

(69794,49058)

(113095,13317)

(61953,8680)

(32571,5476)

(120545,2825)

(146548,2367)

(61100,1832)

(156162,1605)

(19887,1438)

(121531,946)

Section 2E:

Submission command:

```
“spark-submit \
    --class Question2.E \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 10G \
    hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/assignment5_2.11-0.1.jar”
```

I first thoroughly study the documentation for aggregateMessages and the Pregel API in [7], as well as the PDF aggregate messages implementation code in order to understand it.

Afterwards, I copy the aggregateMessages code from the PDF into my code for Part E so I can get a sense of what the output of the code looks like, and develop the commands to sort the output in descending order and then take + display the top 20 results.

Without sorting it, we can see that the output is in the format (Vertex ID, maximum distance from a root). The check value also starts at nearly the same as the number of nodes in the graph and decreases each print until 0, at which point it outputs the edge results as the recursive calls to propagateEdgeCount() resolve. For sorting in descending order and taking the top 20 results, I reuse code from Part D where I sorted by the attribute (number of vertices in community) in descending order as well.

Verbal description of the aggregate messages code:

Each node is initialized to have a distance of 0. Messages are sent to destination that increase the vertex distance of the source by 1 for each step of the node traversed thus far on the way to the destination, each superstep until it reaches the root. Merge message takes all the messages sent to a node during a superstep and takes the highest distance received that round, then directly updates the distance of the vertex. Each round, the previous graph and the new graph are combined together and the difference in distance for each node is detected through subtraction. If

there is a difference, the supersteps continue through recursive calls of `propagateEdgeCount` until the difference is 0 and there is convergence.

Section 2E Final Code:

```

1 package Question2
2 import org.apache.spark._
3 import org.apache.spark.graphx._
4 import org.apache.spark.rdd.RDD
5 import org.apache.spark.SparkContext
6 import org.apache.spark.graphx.GraphLoader
7 import org.apache.spark.graphx.lib.LabelPropagation
8
9 object E {
10 def main(args: Array[String]) {
11
12     // Create Spark Context and load file
13     val sc = new SparkContext()
14     val graph = GraphLoader.edgeListFile(sc, path = "hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/dag_edge_list.txt")
15
16     // Question E queries
17     val initialGraph = graph.mapVertices((_,_) => 0)
18     println("Top 20 nodes with largest distance from root. Format: (Vertex ID, Distance from Root)")
19     val distanceGraph = initialGraph.pregel(initialMsg = 0)(
20         (id,distance,newDistance) => math.max(distance,newDistance), // Vertex program
21         (triplet)=> { // Send Message
22             if(triplet.srcAttr+1 > triplet.dstAttr) {
23                 Iterator((triplet.dstId, triplet.srcAttr+1))
24             } else {
25                 Iterator.empty
26             }
27         },
28         (a:Int,b:Int) => math.max(a,b) // Merge message
29     )
30     distanceGraph.vertices.sortBy(_._2, ascending = false).take( num = 20).foreach(println)
31
32     distanceGraph.vertices.sortBy(_._2, ascending = false).take( num = 20).foreach(println)
33     sc.stop()
34 }
35 }

```

This code heavily draws upon the official GraphX documentation of Pregel [7], where they have an example of using Pregel in the Pregel API section to express shortest path in a single call. There are some differences between this and that code however. Firstly, I had to reverse everything, changing the min function calls to max, and initializing with everything set to 0 instead of positive infinity since we are trying to find the largest distance instead of the smallest. As a result, I also checked if the source attribute + 1 was greater than the destination attribute rather than less. Another difference is that we don't specify a source vertex, but allow for the graph to find the largest distance to a root that may be anywhere.

Section 2E final submission on next page.

Section 2E Final Submission:

Top 20 nodes with largest distance from root. Format: (Vertex ID, Distance from Root)

```
(67660,13)
(162544,13)
(72378,12)
(68972,12)
(76215,12)
(84083,12)
(56685,12)
(114992,11)
(97078,11)
(84516,11)
(39228,11)
(152660,11)
(104386,11)
(129914,11)
(12308,11)
(117900,11)
(112417,11)
(113005,11)
(64969,11)
(129073,11)
```

We can see that the answers when using a single Pregel call are largely identical to those given through the aggregateMessages code in the Assignment 5 PDF. There are minor differences for the 11 distance nodes because there is likely more nodes that are 11 hops away from a root than can fit in the 13 spots available after the 13 and 12 hop nodes. Alternatively, there are exactly thirteen spots, but the program presents them in arbitrary order whenever the vertex distance is tied. In any case, the submission should be sufficiently correct for full marks.

Question 3: Hbase**Section A:**

I first scout out the data we are working with by reviewing Assignment 2, where we used this dataset before. I recall that the values are tab separated with the word, year, number of occurrences, and volume count.

I download the dataset onto the local computer, extract with 7zip, and open with pilot edit to confirm this, then upload it to the cluster with Filezilla.

```
[s1155160788@dicvmd10 Assignment5]$ ls
assignment5_2.11-0.1.jar dag_edge_list.txt edge_list.txt googlebooks-eng-all-1gram-20120701-b.gz mooc_actions.tsv q1a.py q1b.py q1c.py vertices.tsv
[s1155160788@dicvmd10 Assignment5]$

[s1155160788@dicvmd10 Assignment5]$ gzip -d googlebooks-eng-all-1gram-20120701-b.gz
[s1155160788@dicvmd10 Assignment5]$ ls
assignment5_2.11-0.1.jar dag_edge_list.txt edge_list.txt googlebooks-eng-all-1gram-20120701-b mooc_actions.tsv q1a.py q1b.py q1c.py vertices.tsv
[s1155160788@dicvmd10 Assignment5]$
```

I first extract the gzip file with `gzip -d`.

For this question, I primarily refer to the Hbase tutorial [18], the provided references for Import TSV [19], Complete Bulk Load [20], Bulk Load [21], Cloudera on bulk loading [22] and the Apache Hbase book in general [23].

First we need to create the table. We use the following one to create a table with 6 regions, one for each Hregion server minus 1 (just in case one goes unhealthy).

Create Table Command: create 's1155160788','family1', {NUMREGIONS => 6, SPLITALGO => 'HexStringSplit'}

```
hbase(main):001:0> create 's1155160788','family1', {NUMREGIONS => 6, SPLITALGO => 'HexStringSplit'}
0 row(s) in 1.3660 seconds

=> Hbase::Table - s1155160788
hbase(main):002:0>
```

After reading the documentation and the hints, I see that we should have a unique row key for each row. The dataset does not have this by default, as the bigram will appear more than once for each year it is referenced, instead we would want an incrementing number like “1,2,3,4” or incrementing row number like “row1, row2, row3, row4” to be the row key. Although Hbase says a monotonically increasing key is not that good, there is no other easy option for us without writing a MapReduce program custom made for preprocessing the file in the correct way.

After searching and testing, I find [24], a custom command with awk that allows one to prepend the line number, followed by a tab. We can then use this line number as the row key. At this point, I rename the Google book file has googlePrime since we will transform it through commands.

Custom command: awk '{printf "%d\t%s\n", NR, \$0}' < fileName > newFileName

Below is a screenshot of sample output from this command.

1	B'enard	1974	1	1
2	B'enard	1982	1	1
3	B'enard	1993	1	1
4	B'enard	1997	1	1
5	B'enard	2001	2	1
6	B'enard	2003	3	2
7	B'enard	2004	8	6
8	B'enard	2005	25	6
9	B'enard	2006	156	14
10	B'enard	2007	159	19
11	B'enard	2008	44	18
12	B'h_NOUN	1794	1	1

I tried running this on the Google book file using: awk '{printf "%d\t%s\n", NR, \$0}' < googlePrime > googleDataset

```
awk: cmd. line:1: (FILENAME== FNR=59816587) fatal: printf to "standard output" failed (No space left on device)
```

The file is 61,551,917 lines long. Unfortunately, the local machine runs out of space printing to standard output around line 60 million, just short of finishing. To resolve this, I download and install Gawk32, a program that allows for the use of awk in Windows command line. I need to add the bin folder to the path variable of my Windows machine. Afterwards, it is not enough to use the Linux command. According to [25], when using awk on Windows through gawk, you need to use double quotes (“”) where there would otherwise be single quotes (‘), and if there are double quotes within the command, you need to use (\\”) to escape it.

Windows equivalent: `awk "{printf \"%d\t%s\n\", NR, $0}" < googlePrime > googleDataset`

```
C:\Users\Eddie\Documents\School\CUHK\2021 - 2022 Term 2\IEHS 5730 - Big Data Systems and Information Processing\Week 16\Assignment 5>awk "{printf \"%d\t%s\n\", NR, $0}" < googlePrime > googleDataset
C:\Users\Eddie\Documents\School\CUHK\2021 - 2022 Term 2\IEHS 5730 - Big Data Systems and Information Processing\Week 16\Assignment 5>
```

On my local machine, I am able to run this command without running out of space to create an approximately 1.8 GB file called `googleDataset`, that I then transfer over to the IE DIC cluster. I confirm through “`wc -l`” that the line count is the same and through `cat | less` that the format is perfect, with the line number being tab separated from the rest of the original dataset, able to act as the row key.

```
1480    B48_ADJ 1999    14    5
1481    B48_ADJ 2000    4    3
1482    B48_ADJ 2001    5    5
1483    B48_ADJ 2002    14    9
1484    B48_ADJ 2003    5    4
1485    B48_ADJ 2004    3    3
1486    B48_ADJ 2005    15    14
1487    B48_ADJ 2006    10    10
1488    B48_ADJ 2007    84    13
1489    B48_ADJ 2008    9    9
1490    B5R      1922    1    1
1491    B5R      1931    1    1
1492    B5R      1942    1    1
```

```
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -copyFromLocal googleDataset
[s1155160788@dicvmd10 Assignment5]$ rm googleDataset
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -ls
Found 7 items
drwxr-xr-x  - s1155160788 student          0 2022-05-09 22:12 GoogleStore
-rw-r--r--  3 s1155160788 student      37636 2022-05-09 16:44 assignment5_2.11-0.1.jar
-rw-r--r--  3 s1155160788 student    2142233 2022-05-08 15:53 dag_edge_list.txt
-rw-r--r--  3 s1155160788 student    14773350 2022-05-08 15:53 edge_list.txt
-rw-r--r--  3 s1155160788 student 1872801001 2022-05-10 01:47 googleDataset
-rw-r--r--  3 s1155160788 student    7996988 2022-05-07 10:40 mooc_actions.tsv
-rw-r--r--  3 s1155160788 student     71405 2022-05-07 10:40 vertices.tsv
[s1155160788@dicvmd10 Assignment5]$
```

Finally, I copy the `googleDataset` to the HDFS and create a directory called `GoogleStore` for the storage files later, and erase `googleDataset` from the local machine to save space for others, since it is in low supply.

HDFS storage directory: `hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/GoogleStore`

Google Dataset file path: `hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/googleDataset`

ImportTSV command: hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
 Dimporrtsv.bulk.output=/user/s1155160788/GoogleStore/googleHfile -
 Dimporrtsv.columns=HBASE_ROW_KEY,family1:year,family1:match_count,f
 amily1:volume_count s1155160788 /user/s1155160788/googleDataset

```
[s1155160788@dicvmd10 Assignment5]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporrtsv.bulk.output=/user/s1155160788/GoogleStore/googleHfile -Dimporrtsv.columns=HBASE_ROW_KEY,fam
ily1:bigram,family1:year,family1:match_count,family1:volume_count s1155160788 /user/s1155160788/googleDataset
2022-05-10 12:57:31,563 INFO [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x6a6afff2 connecting to ZooKeeper ensemble=dicvmd1.ie.cuhk.edu.hk:2181,dicvmd2.ie.cuhk.e
du.hk:2181,dicvmd3.ie.cuhk.edu.hk:2181,dicvmd7.ie.cuhk.edu.hk:2181
2022-05-10 12:57:31,570 INFO [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-292--1, built on 05/11/2018 07:15 GMT
2022-05-10 12:57:31,570 INFO [main] zookeeper.ZooKeeper: Client environment:host.name=dicvmd10.ie.cuhk.edu.hk
2022-05-10 12:57:31,570 INFO [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_112
2022-05-10 12:57:31,570 INFO [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2022-05-10 12:57:31,570 INFO [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/jdk64/jdk1.8.0_112/jre
2022-05-10 12:57:31,571 INFO [main] zookeeper.ZooKeeper: Client environment:java.class.path=/usr/hdp/2.6.5.0-292/hbase/conf:/usr/jdk64/jdk1.8.0_112/lib/tools.jar:/usr/hdp/2.6.5.0-292/hbase
:/usr/hdp/2.6.5.0-292/hbase/lib/activation-1.1.jar:/usr/hdp/2.6.5.0-292/hbase/lib/aopalliance-1.0.jar:/usr/hdp/2.6.5.0-292/hbase/lib/apacheds-118n-2.0.0-M15.jar:/usr/hdp/2.6.5.0-292/hbase/l
ib/apacheds-kerberos-codec-2.0.0-M15.jar:/usr/hdp/2.6.5.0-292/hbase/lib/api-asn1-api-1.0.0-M20.jar:/usr/hdp/2.6.5.0-292/hbase/lib/api-util-1.0.0-M20.jar:/usr/hdp/2.6.5.0-292/hbase/lib/asm-3
.1.jar:/usr/hdp/2.6.5.0-292/hbase/lib/avro-1.7.6.jar:/usr/hdp/2.6.5.0-292/hbase/lib/avx-java-sdk-core-1.10.6.jar:/usr/hdp/2.6.5.0-292/hbase/lib/avx-java-sdk-kms-1.10.6.jar:/usr/hdp/2.6.5.0-
```

Output omitted.

```
2022-05-10 13:08:45,198 INFO [main] mapreduce.Job: map 100% reduce 100%
2022-05-10 13:08:50,211 INFO [main] mapreduce.Job: Job job_1650268761049_3237 completed successfully
2022-05-10 13:08:50,286 INFO [main] mapreduce.Job: Counters: 50
  File System Counters
    FILE: Number of bytes read=9432573966
    FILE: Number of bytes written=18867995559
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=1874506757
    HDFS: Number of bytes written=12456685250
    HDFS: Number of read operations=49
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=4
  Job Counters
    Launched map tasks=14
    Launched reduce tasks=1
    Data-local map tasks=14
    Total time spent by all maps in occupied slots (ms)=910118
    Total time spent by all reduces in occupied slots (ms)=1761168
    Total time spent by all map tasks (ms)=455059
    Total time spent by all reduce tasks (ms)=440292
    Total vcore-milliseconds taken by all map tasks=455059
    Total vcore-milliseconds taken by all reduce tasks=440292
    Total megabyte-milliseconds taken by all map tasks=14911373312
    Total megabyte-milliseconds taken by all reduce tasks=28854976512
  Map-Reduce Framework
    Map input records=61551917
    Map output records=61551917
    Map output bytes=9247919419
    Map output materialized bytes=9432574044
    Input split bytes=1820
    Combine input records=61551917
    Combine output records=61551917
    Reduce input groups=61551917
    Reduce shuffle bytes=9432574044
```

```

ImportTsv
    Bad Lines=0
Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=1874504937
File Output Format Counters
    Bytes Written=12456685250
[s1155160788@dicvmd10 Assignment5]$

```

```

[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -ls
Found 9 items
drwxr-xr-x - s1155160788 student 0 2022-05-10 13:08 .staging
drwxr-xr-x - s1155160788 student 0 2022-05-10 12:57 GoogleStore
-rw-r--r-- 3 s1155160788 student 37636 2022-05-09 16:44 assignment5_2.11-0.1.jar
-rw-r--r-- 3 s1155160788 student 2142233 2022-05-08 15:53 dag_edge_list.txt
-rw-r--r-- 3 s1155160788 student 14773350 2022-05-08 15:53 edge_list.txt
-rw-r--r-- 3 s1155160788 student 1872801001 2022-05-10 01:47 googleDataset
drwxr-xr-x - s1155160788 student 0 2022-05-10 13:08 hbase-staging
-rw-r--r-- 3 s1155160788 student 7996988 2022-05-07 10:40 mooc_actions.tsv
-rw-r--r-- 3 s1155160788 student 71405 2022-05-07 10:40 vertices.tsv
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -ls GoogleStore
Found 1 items
drwxr-xr-x - s1155160788 student 0 2022-05-10 13:08 GoogleStore/googleHfile
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -ls GoogleStore/googleHfile
Found 2 items
-rw-r--r-- 3 s1155160788 student 0 2022-05-10 13:08 GoogleStore/googleHfile/_SUCCESS
drwxr-xr-x - s1155160788 student 0 2022-05-10 13:08 GoogleStore/googleHfile/family1
[s1155160788@dicvmd10 Assignment5]$ hdfs dfs -ls GoogleStore/googleHfile/family1
Found 2 items
-rw-r--r-- 3 s1155160788 student 953407341 2022-05-10 13:08 GoogleStore/googleHfile/family1/9fcd99a1604f402dbd73af5773cb2d0f
-rw-r--r-- 3 s1155160788 student 11503277909 2022-05-10 13:08 GoogleStore/googleHfile/family1/dd9cfa8e82f54e47bda523dcbbaee6af
[s1155160788@dicvmd10 Assignment5]$

```

We can see that there are two hfiles uploaded in HDFS under GoogleStore/googleHfile/family1. I thought that there would only be one hfile, which is why I specified googleHfile, but it created a directory for that and then presumably there is one folder for each column family.

Complete bulk command: `hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/GoogleStore/googleHfile s1155160788`

```

[s1155160788@dicvmd10 Assignment5]$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/GoogleStore/googleHfile s1155160788
2022-05-10 13:29:48,887 INFO [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x57d7f8ca connecting to ZooKeeper ensemble=dicvmd1.ie.cuhk.edu.hk:2181,dicvmd2.ie.cuhk.edu.hk:2181,dicvmd3.ie.cuhk.edu.hk:2181
2022-05-10 13:29:48,894 INFO [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-292-1, built on 05/11/2018 07:15 GMT
2022-05-10 13:29:48,894 INFO [main] zookeeper.ZooKeeper: Client environment:host.name=dicvmd10.ie.cuhk.edu.hk
2022-05-10 13:29:48,894 INFO [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_112
2022-05-10 13:29:48,894 INFO [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2022-05-10 13:29:48,894 INFO [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/jdk64/jdk1.8.0_112/jre
2022-05-10 13:29:48,894 INFO [main] zookeeper.ZooKeeper: Client environment:java.class.path=/usr/hdp/2.6.5.0-292/hbase/conf:/usr/jdk64/jdk1.8.0_112/lib/tools.jar:/usr/hdp/2.6.5.0-292/hbase:/usr/hdp/2.6.5.0-292/hbase/lib/activation-1.1.jar:/usr/hdp/2.6.5.0-292/hbase/lib/activation-1.0.jar:/usr/hdp/2.6.5.0-292/hbase/lib/apacheds-115n-2.0.0-M15.jar:/usr/hdp/2.6.5.0-292/hbase/lib/

```

Output omitted.

```

2022-05-10 13:29:49,982 WARN [main] mapreduce.LoadIncrementalHFiles: Skipping non-directory hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/GoogleStore/googleHfile/_SUCCESS
2022-05-10 13:29:50,239 INFO [main] impl.MetricsConfig: loaded properties from hadoop-metrics2-hbase.properties
2022-05-10 13:29:50,349 INFO [main] timeline.HadoopTimelineMetricsSink: Initializing Timeline metrics sink.
2022-05-10 13:29:50,392 INFO [main] timeline.HadoopTimelineMetricsSink: Identified hostname = dicvmd10.ie.cuhk.edu.hk, serviceName = hbase
2022-05-10 13:29:50,404 INFO [main] availability.MetricSinkWriteShardHostnameHashingStrategy: Calculated collector shard dicvmd4.ie.cuhk.edu.hk based on hostname: dicvmd10.ie.cuhk.edu.hk
2022-05-10 13:29:50,404 INFO [main] timeline.HadoopTimelineMetricsSink: Collector Uri: http://dicvmd4.ie.cuhk.edu.hk:6188/ws/v1/timeline/metrics
2022-05-10 13:29:50,404 INFO [main] timeline.HadoopTimelineMetricsSink: Container Metrics Uri: http://dicvmd4.ie.cuhk.edu.hk:6188/ws/v1/timeline/containermetrics
2022-05-10 13:29:50,412 INFO [main] impl.MetricsSinkAdapter: Sink timeline started
2022-05-10 13:29:50,468 INFO [main] impl.MetricsSystemImpl: Scheduled snapshot period at 10 second(s).
2022-05-10 13:29:50,468 INFO [main] impl.MetricsSystemImpl: Hbase metrics system started
2022-05-10 13:29:50,765 INFO [LoadIncrementalHFiles-0] hfile.CacheConfig: CacheConfig:disabled
2022-05-10 13:29:50,765 INFO [LoadIncrementalHFiles-0] hfile.CacheConfig: CacheConfig:disabled
2022-05-10 13:29:50,765 INFO [LoadIncrementalHFiles-1] hfile.CacheConfig: CacheConfig:disabled
2022-05-10 13:29:50,765 INFO [LoadIncrementalHFiles-3] hfile.CacheConfig: CacheConfig:disabled
2022-05-10 13:29:50,765 INFO [LoadIncrementalHFiles-2] hfile.CacheConfig: CacheConfig:disabled
2022-05-10 13:29:50,832 INFO [LoadIncrementalHFiles-0] mapreduce.LoadIncrementalHFiles: Trying to load hfile=hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/GoogleStore/googleHfile/family1/2f7e713cbbf4702a01076781791c290 first=0 last=9999999
2022-05-10 13:29:50,832 INFO [LoadIncrementalHFiles-3] mapreduce.LoadIncrementalHFiles: Trying to load hfile=hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/GoogleStore/googleHfile/family1/e884357f3df44853886eb3023f038a7c first=5555554 last=7999999
2022-05-10 13:29:50,832 INFO [LoadIncrementalHFiles-2] mapreduce.LoadIncrementalHFiles: Trying to load hfile=hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/GoogleStore/googleHfile/family1/fad73497a6d447a66197b1623ee42d first=3 last=5555553
2022-05-10 13:29:50,832 INFO [LoadIncrementalHFiles-1] mapreduce.LoadIncrementalHFiles: Trying to load hfile=hdfs://dicvmd2.ie.cuhk.edu.hk:8020/user/s1155160788/GoogleStore/googleHfile/family1/5347ac562487433c8e71996e473153ef first=1 last=2999999
2022-05-10 13:29:50,929 INFO [main] client.ConnectionManager$HConnectionImplementation: Closing master protocol: MasterService
2022-05-10 13:29:50,930 INFO [main] client.ConnectionManager$HConnectionImplementation: Closing zookeeper sessionid=0x17e77b8983a0306
2022-05-10 13:29:50,932 INFO [main] zookeeper.ZooKeeper: Session: 0x17e77b8983a0306 closed
2022-05-10 13:29:50,932 INFO [main-EventThread] zookeeper.ClientCnxn: EventThread shut down
[s1155160788@dicvmd10 Assignment$1]
[s1155160788@dicvmd10 ~]$ hdfs dfs -ls GoogleStore
Found 1 items
drwxr-xr-x - s1155160788 student 0 2022-05-10 13:34 GoogleStore/googleHfile
[s1155160788@dicvmd10 ~]$ hdfs dfs -ls GoogleStore/googleHfile
Found 1 items
drwxr-xr-x - s1155160788 student 0 2022-05-10 13:29 GoogleStore/googleHfile/family1
[s1155160788@dicvmd10 ~]$ hdfs dfs -ls GoogleStore/googleHfile/family1
[s1155160788@dicvmd10 ~]$

```

It appears that after running the bulk load command, and the rows are loaded into the table, the Hfiles are deleted.

```

Current count: 61550000, row: 9998272
Current count: 61551000, row: 9999172
61551917 row(s) in 2536.2400 seconds

=> 61551917
hbase(main):002:0>

```

I ran a: count “ ‘s1155160788’ ” command to confirm that every row has been added, and it is exactly equal to the number of lines in the dataset.

Section 3B:

Subsection 1:

To insert a new row, I will need to use a new row key. Since my row keys are set up to be the line number of the record, I need to use a row key equal to the line number of the dataset +1 for a new row.

This is $61551917 + 1 = 61551918$

We also need to keep in mind that we can only put the value for one column of the row at a time, so we need 4 put commands.

Put / Insert commands:

Syntax (this is not a command I used): put 'table_name', 'row_key', 'column_name', 'value'
 put 's1155160788', '61551918', 'bigram', 'ierg4330'
 put 's1155160788', '61551918', 'year', '2019'
 put 's1155160788', '61551918', 'match_count', '100'

```

put 's1155160788', '61551918', 'volume_count', '4'
hbase(main):002:0> put 's1155160788', '61551918', 'family1:bigram', 'ierg4330'
0 row(s) in 0.1320 seconds

hbase(main):003:0> put 's1155160788', '61551918', 'family1:year', '2019'
0 row(s) in 0.0540 seconds

hbase(main):004:0> put 's1155160788', '61551918', 'family1:match_count', '100'
0 row(s) in 0.0200 seconds

hbase(main):005:0> put 's1155160788', '61551918', 'family1:volume_count', '4'
0 row(s) in 0.0300 seconds

hbase(main):006:0> █

```

Get Command to Verify results: get 's1155160788', '61551918'

```

hbase(main):006:0> get 's1155160788', '61551918'
COLUMN                                CELL
family1:bigram                        timestamp=1652163769126, value=ierg4330
family1:match_count                   timestamp=1652163781609, value=100
family1:volume_count                  timestamp=1652163787115, value=4
family1:year                          timestamp=1652163775544, value=2019
4 row(s) in 0.0510 seconds

hbase(main):007:0> █

```

Subsection 2:

Here I refer to [23], [26], and [27] documentation for creating the scan + filtering commands. I struggled greatly with this section due to some nuances but managed to succeed in the end.

My initial scan command attempt was something like the following:

```

scan 's1155160788', { FILTER=>"SingleColumnValueFilter('family1', 'year', =, 'binary:1671')
AND SingleColumnValueFilter('family1', 'match_count', >, 'binary:100')"}

```

The year filter was working correctly, but it seemed that the filter on the match_count column was being ignored. Through looking at the Blackboard forums and the TA comments, it seems that SingleColumnValueFilter uses the binary comparator by default and this does lexicographic comparison. The value 100 is not stored as an integer or other numerical type but as a byte array. The comparisons take place character by character. A number like 37 would be considered greater than 100 in this comparison system because 3 is greater than 1.

Second scan command attempt based on [28]:

```

import java.lang.Long
import org.apache.hadoop.hbase.util.Bytes
import org.apache.hadoop.hbase.filter.SingleColumnValueFilter
import org.apache.hadoop.hbase.filter.CompareFilter
import org.apache.hadoop.hbase.filter.BinaryComparator

```

```
scan "s1155160788", { FILTER => SingleColumnValueFilter.new(Bytes.toBytes('family1'),
Bytes.toBytes('match_count'), CompareFilter::CompareOp.valueOf('GREATER'),
BinaryComparator.new(Bytes.toBytes(Long.parseLong("100"))))}}
```

This also failed because even though the scan went through, the match_count filter was still being ignored.

I thought about what I knew. I knew that the equal comparator was functional, so I needed to do some processing that would allow the format of the value on the right side of the comparator to be a triple digit number. [29] opened my eyes to the potential of using regexstrings on the right hand side to force the scan to only allow for match_count values that equal a regexstring. I reviewed regex at [30] to devise the following and final scan command.

Scan command: scan 's1155160788', { FILTER=>"SingleColumnValueFilter('family1', 'year', =, 'binary:1671') AND SingleColumnValueFilter('family1', 'match_count', =, 'regexstring:^(1-9)[0-9]{2,})'" }

^[1-9] means the first character needs to be 1-9. [0-9]{2,} means the first digit is followed by 2 or more numbers between 0-9. This effectively forces the scan to filter for match_counts that are 3 digit numbers The first number being forced to 1-9 prevents zero padded numbers from erroneously being perceived as over 100, such as 007.

Scan output on next page.

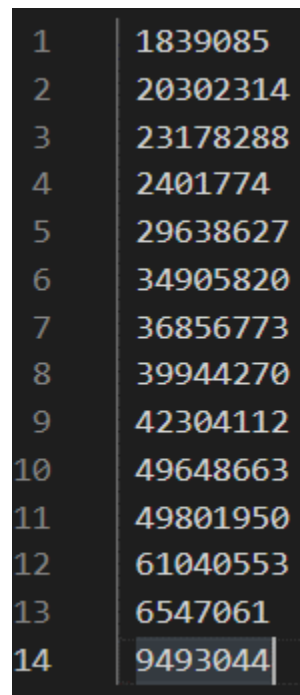

```

ROW          COLUMN+CELL
1839085      column=family1:bigram, timestamp=1652159829147, value=but
1839085      column=family1:match_count, timestamp=1652159829147, value=644
1839085      column=family1:volume_count, timestamp=1652159829147, value=4
1839085      column=family1:year, timestamp=1652159829147, value=1671
20302314     column=family1:bigram, timestamp=1652159829147, value=but_CONJ
20302314     column=family1:match_count, timestamp=1652159829147, value=643
20302314     column=family1:volume_count, timestamp=1652159829147, value=4
20302314     column=family1:year, timestamp=1652159829147, value=1671
23178288     column=family1:bigram, timestamp=1652159829147, value=been
23178288     column=family1:match_count, timestamp=1652159829147, value=237
23178288     column=family1:volume_count, timestamp=1652159829147, value=4
23178288     column=family1:year, timestamp=1652159829147, value=1671
2401774      column=family1:bigram, timestamp=1652159829147, value=be
2401774      column=family1:match_count, timestamp=1652159829147, value=1230
2401774      column=family1:volume_count, timestamp=1652159829147, value=4
2401774      column=family1:year, timestamp=1652159829147, value=1671
29638627     column=family1:bigram, timestamp=1652159829147, value=being
29638627     column=family1:match_count, timestamp=1652159829147, value=224
29638627     column=family1:volume_count, timestamp=1652159829147, value=4
29638627     column=family1:year, timestamp=1652159829147, value=1671
34905820     column=family1:bigram, timestamp=1652159829147, value=by
34905820     column=family1:match_count, timestamp=1652159829147, value=816
34905820     column=family1:volume_count, timestamp=1652159829147, value=4
34905820     column=family1:year, timestamp=1652159829147, value=1671
36856773     column=family1:bigram, timestamp=1652159829147, value=But_CONJ
36856773     column=family1:match_count, timestamp=1652159829147, value=158
36856773     column=family1:volume_count, timestamp=1652159829147, value=4
36856773     column=family1:year, timestamp=1652159829147, value=1671
39944270     column=family1:bigram, timestamp=1652159829147, value=being VERB
39944270     column=family1:match_count, timestamp=1652159829147, value=196
39944270     column=family1:volume_count, timestamp=1652159829147, value=4
39944270     column=family1:year, timestamp=1652159829147, value=1671
42304112     column=family1:bigram, timestamp=1652159829147, value=by ADP
42304112     column=family1:match_count, timestamp=1652159829147, value=805
42304112     column=family1:volume_count, timestamp=1652159829147, value=4
42304112     column=family1:year, timestamp=1652159829147, value=1671
49648663     column=family1:bigram, timestamp=1652159829147, value=because
49648663     column=family1:match_count, timestamp=1652159829147, value=137
49648663     column=family1:volume_count, timestamp=1652159829147, value=4
49648663     column=family1:year, timestamp=1652159829147, value=1671
49801950     column=family1:bigram, timestamp=1652159829147, value=be VERB
49801950     column=family1:match_count, timestamp=1652159829147, value=1227
49801950     column=family1:volume_count, timestamp=1652159829147, value=4
49801950     column=family1:year, timestamp=1652159829147, value=1671
61040553     column=family1:bigram, timestamp=1652159829147, value=because ADP
61040553     column=family1:match_count, timestamp=1652159829147, value=137
61040553     column=family1:year, timestamp=1652159829147, value=1671
6547061      column=family1:bigram, timestamp=1652159829147, value=But
6547061      column=family1:match_count, timestamp=1652159829147, value=158
6547061      column=family1:volume_count, timestamp=1652159829147, value=4
6547061      column=family1:year, timestamp=1652159829147, value=1671
9493044      column=family1:bigram, timestamp=1652159829147, value=been VERB
9493044      column=family1:match_count, timestamp=1652159829147, value=237
9493044      column=family1:volume_count, timestamp=1652159829147, value=4
9493044      column=family1:year, timestamp=1652159829147, value=1671
14 row(s) in 80.5650 seconds

hbase(main):002:0>

```

By copyasting the output into a text file and processing it, we can get the 14 row keys with match_count greater than 100, verified by manual inspection in the screenshots above.



1	1839085
2	20302314
3	23178288
4	2401774
5	29638627
6	34905820
7	36856773
8	39944270
9	42304112
10	49648663
11	49801950
12	61040553
13	6547061
14	9493044

Delete command: `deleteall 'table_name','row_key'`
Screenshot of deletion on the next page.


```
hbase(main):001:0> deleteall 's1155160788','1839085'  
0 row(s) in 0.1820 seconds  
  
hbase(main):002:0> deleteall 's1155160788','20302314'  
0 row(s) in 0.0150 seconds  
  
hbase(main):003:0> deleteall 's1155160788','23178288'  
deleteall 's1155160788','6547061'  
deleteall 's1155160788','9493044' 0 row(s) in 0.0120 seconds  
  
hbase(main):004:0> deleteall 's1155160788','2401774'  
0 row(s) in 0.0040 seconds  
  
hbase(main):005:0> deleteall 's1155160788','29638627'  
0 row(s) in 0.0050 seconds  
  
hbase(main):006:0> deleteall 's1155160788','34905820'  
0 row(s) in 0.0120 seconds  
  
hbase(main):007:0> deleteall 's1155160788','36856773'  
0 row(s) in 0.0040 seconds  
  
hbase(main):008:0> deleteall 's1155160788','39944270'  
0 row(s) in 0.0050 seconds  
  
hbase(main):009:0> deleteall 's1155160788','42304112'  
0 row(s) in 0.0090 seconds  
  
hbase(main):010:0> deleteall 's1155160788','49648663'  
0 row(s) in 0.0040 seconds  
  
hbase(main):011:0> deleteall 's1155160788','49801950'  
0 row(s) in 0.0030 seconds  
  
hbase(main):012:0> deleteall 's1155160788','61040553'  
0 row(s) in 0.0130 seconds  
  
hbase(main):013:0> deleteall 's1155160788','6547061'  
0 row(s) in 0.0030 seconds  
  
hbase(main):014:0> deleteall 's1155160788','9493044'  
0 row(s) in 0.0210 seconds  
  
hbase(main):015:0> █
```

```

hbase(main):016:0> get 's1155160788','9493044'
COLUMN                                CELL
0 row(s) in 0.0340 seconds

hbase(main):017:0> █

```

Calling get on any of the 14 row keys indicated in the scan now shows the above. We can see that the 14 deletions went through successfully.

References

- [1] https://mobitec.ie.cuhk.edu.hk/iems5730/tutorials/10_kafka_k8s/
- [2] https://graphframes.github.io/graphframes/docs/_site/user-guide.html
- [3] <https://sparkbyexamples.com/pyspark/pyspark-read-csv-file-into-dataframe/>
- [4] <https://stackoverflow.com/questions/39541204/pyspark-nameerror-name-spark-is-not-defined>
- [5] <https://sparkbyexamples.com/pyspark/pyspark-rename-dataframe-column/>
- [6] <https://towardsdatascience.com/graphframes-in-jupyter-a-practical-guide-9b3b346cebc5>
- [7] <https://spark.apache.org/docs/2.3.0/graphx-programming-guide.html>
- [8] <https://stackoverflow.com/questions/55039384/how-printf-works-in-scala>
- [9] <https://stackoverflow.com/questions/47170349/number-of-connected-components-in-a-directed-graph-in-graphx>
- [10] <https://livebook.manning.com/book/spark-in-action/chapter-9/153>
- [11] <https://github.com/amplab/graphx/blob/master/graphx/src/main/scala/org/apache/spark/graphx/lib/StronglyConnectedComponents.scala>
- [12] <https://livebook.manning.com/book/spark-in-action/chapter-9/161>
- [13] <https://github.com/apache/spark/blob/master/graphx/src/main/scala/org/apache/spark/graphx/GraphOps.scala>
- [14] <https://stackoverflow.com/questions/36963319/spark-rdd-sort-by-two-values>
- [15] <https://stackoverflow.com/questions/59763659/pyspark-retrieve-first-element-of-rdd-top1-vs-first>
- [16] <https://github.com/apache/spark/blob/master/graphx/src/main/scala/org/apache/spark/graphx/lib/LabelPropagation.scala>

- [17] <https://stackoverflow.com/questions/44644685/how-to-sort-spark-countbykey-result-which-is-in-scala-collection-mapstring>
- [18] https://mobitec.ie.cuhk.edu.hk/iems5730/tutorials/11_hbase/
- [19] <https://hbase.apache.org/book.html#importtsv>
- [20] <https://hbase.apache.org/book.html#completebulkload>
- [21] <https://hbase.apache.org/book.html#arch.bulk.load>
- [22] https://docs.cloudera.com/documentation/enterprise/6/6.1/topics/admin_hbase_import.html#concept_mnj_psz_wp
- [23] <https://hbase.apache.org/book.html>
- [24] <https://superuser.com/questions/10201/how-can-i-prepend-a-line-number-and-tab-to-each-line-of-a-text-file>
- [25] <https://stackoverflow.com/questions/4852270/grep-and-awk-in-windows-invalid-char-in-expression-error>
- [26] <https://sparkbyexamples.com/hbase/hbase-table-filtering-data-like-where-clause/>
- [27] https://docs.cloudera.com/documentation/enterprise/6/6.3/topics/admin_hbase_filtering.html
- [28] <https://stackoverflow.com/questions/27632835/how-to-use-filters-over-atomic-counter-in-hbase>
- [29] <https://stackoverflow.com/questions/56766332/rowfilter-with-scan-query-in-the-hbase>
- [30] <https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285>