Write a brief report about your findings, answering the following questions:

Link to Output Logs and Files

1. [10 points] Show the pseudo-code for all 5 Twitter-follower-count programs (RDD-G, RDD-R, RDD-F, RDD-A, DSET) in Spark Scala. Since many Scala functions are similar to pseudo-code, you may copy-and-paste well-designed (good variable naming!) and well-commented Scala code fragments here.

RDD-G

```
// Read the input text file and create an RDD of (userID, followerID)
   val textFile = sc.textFile(args(0))
   val userNumOfFollowersRDD = textFile.map(line => line.split(","))
                              .map(fields => (fields(1).toInt,fields(0).toInt))
                              .filter { case (userId, _) => userId % 100 == 0 }
                              .groupByKey()
                              .mapValues(_.size)

   // Print the debug string for groupedRDD
   println("Debug String for userNumOfFollowersRDD:")
   println(userNumOfFollowersRDD.toDebugString)

   userNumOfFollowersRDD.saveAsTextFile(args(1))
```

RDD-R

```
val textFile = sc.textFile(args(0))
   val userNumOfFollowersRDD = textFile.map(line => line.split(","))
                              .map(fields => (fields(1).toInt,1))
                              .filter { case (userId, _) => userId % 100 == 0 }   // Filter the RDD to
include only users with at least one follower and whose ID is divisible by 100
                              .reduceByKey(_+_)

   // Print the debug string for groupedRDD
   println("Debug String for userNumOfFollowersRDD:")
   println(userNumOfFollowersRDD.toDebugString)

   // Save the result to the output directory
   userNumOfFollowersRDD.saveAsTextFile(args(1))
```

RDD-F

```
val textFile = sc.textFile(args(0))
   val userNumOfFollowersRDD = textFile.map(line => line.split(","))
                              .map(fields => (fields(1).toInt,1))
```

```
                        .filter { case (userId, _) => userId % 100 == 0 }   // Filter the RDD to
include only users with at least one follower and whose ID is divisible by 100
                        .foldByKey(0)(_+_)

    // Print the debug string for groupedRDD
    println("Debug String for userNumOfFollowersRDD:")
    println(userNumOfFollowersRDD.toDebugString)

    // Save the result to the output directory
    userNumOfFollowersRDD.saveAsTextFile(args(1))
```

RDD-A

```
 // Read the input text file and create an RDD of (userID, followerID)
    val textFile = sc.textFile(args(0))
    val userNumOfFollowersRDD = textFile.map(line => line.split(","))
                        .map(fields => (fields(1).toInt,1))
                        .filter { case (userId, _) => userId % 100 == 0 }   // Filter the RDD to
include only users with at least one follower and whose ID is divisible by 100
                        .aggregateByKey(0)(
                        (accumulator: Int, value: Int) => accumulator + value,
                        (accumulator1: Int, accumulator2: Int) => accumulator1 + accumulator2 )



    // Print the debug string for groupedRDD
    println("Debug String for userNumOfFollowersRDD:")
    println(userNumOfFollowersRDD.toDebugString)
```

DSET

```
// Read input CSV file into DataFrame
    val df: DataFrame = spark.read
      .format("csv")
      .option("inferSchema", "true") // Infer schema from data types
      .load(args(0))
      .toDF("follower", "followee") // Specify column names

    df.explain()

    // Filter rows where the value of the "followee" column is divisible by 100, then group by
"followee" and count occurrences
    import spark.implicits._
    val countDF = df.filter($"followee" % 100 === 0).groupBy("followee").count()

    // Save the result to the output directory
    countDF.write.format("csv").save(args(1))
```

2. [10 points] Show the link(s) to the source code for these programs in your Github Classroom repository.

RDD-G:
https://github.com/CS6240/hw-3-AkshaySyal/blob/master/src/main/scala/wc/RDDG.scala

RDD-R:
https://github.com/CS6240/hw-3-AkshaySyal/blob/master/src/main/scala/wc/RDDR.scala

RDD-F:
https://github.com/CS6240/hw-3-AkshaySyal/blob/master/src/main/scala/wc/RDDF.scala

RDD-A:
https://github.com/CS6240/hw-3-AkshaySyal/blob/master/src/main/scala/wc/RDDA.scala

DSET: https://github.com/CS6240/hw-3-AkshaySyal/blob/master/src/main/scala/wc/DSET.scala

3. [10 points] Run these programs locally (not on AWS) and determine if Spark performs aggregation before data is shuffled, i.e., the equivalent of MapReduce's Combiner. Look for evidence that supports your answer, e.g., by using toDebugString() and explain(), looking at the log files, and consulting the Spark documentation. For each of the 5 programs, (1) state clearly if it aggregates before shuffling (like a Combiner) and (2) present evidence to support your answer.

Log for RDD-G
Debug String for userNumOfFollowersRDD:
(40) MapPartitionsRDD[6] at mapValues at RDDG.scala:29 []
 | ShuffledRDD[5] at groupByKey at RDDG.scala:27 []
 +-(40) MapPartitionsRDD[4] at filter at RDDG.scala:27 []
    | MapPartitionsRDD[3] at map at RDDG.scala:26 []
    | MapPartitionsRDD[2] at map at RDDG.scala:25 []
    | input MapPartitionsRDD[1] at textFile at RDDG.scala:24 []
    | input HadoopRDD[0] at textFile at RDDG.scala:24 []

Log of RDD-R
Debug String for userNumOfFollowersRDD:
(40) ShuffledRDD[5] at reduceByKey at RDDR.scala:28 []
 +-(40) MapPartitionsRDD[4] at filter at RDDR.scala:27 []
    | MapPartitionsRDD[3] at map at RDDR.scala:26 []
    | MapPartitionsRDD[2] at map at RDDR.scala:25 []
    | input MapPartitionsRDD[1] at textFile at RDDR.scala:24 []
    | input HadoopRDD[0] at textFile at RDDR.scala:24 []

Log of RDD-F
Debug String for userNumOfFollowersRDD:
(40) ShuffledRDD[5] at foldByKey at RDDF.scala:28 []
 +-(40) MapPartitionsRDD[4] at filter at RDDF.scala:27 []
    | MapPartitionsRDD[3] at map at RDDF.scala:26 []
    | MapPartitionsRDD[2] at map at RDDF.scala:25 []
    | input MapPartitionsRDD[1] at textFile at RDDF.scala:24 []
    | input HadoopRDD[0] at textFile at RDDF.scala:24 []

Log of RDD-A
Debug String for userNumOfFollowersRDD:
(40) ShuffledRDD[5] at aggregateByKey at RDDA.scala:28 []
 +-(40) MapPartitionsRDD[4] at filter at RDDA.scala:27 []
    | MapPartitionsRDD[3] at map at RDDA.scala:26 []
    | MapPartitionsRDD[2] at map at RDDA.scala:25 []
    | input MapPartitionsRDD[1] at textFile at RDDA.scala:24 []
    | input HadoopRDD[0] at textFile at RDDA.scala:24 []

Log of DSET
== Physical Plan ==
*(1) Project [_c0#17 AS follower#21, _c1#18 AS followee#22]
+- FileScan csv [_c0#17,_c1#18] Batched: false, DataFilters: [], Format: CSV, Location: InMemoryFileIndex(1 paths)[file:/app/hw1-spark-AkshaySyal-main/input], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<_c0:int,_c1:int>

Relevant log output:
2024-02-13 02:49:14,191 INFO scheduler.DAGScheduler: Got map stage job 2 (save at DSET.scala:34) with 10 output partitions
2024-02-13 02:49:14,192 INFO scheduler.DAGScheduler: Final stage: ShuffleMapStage 2 (save at DSET.scala:34)
2024-02-13 02:49:14,193 INFO scheduler.DAGScheduler: Parents of final stage: List()
2024-02-13 02:49:14,196 INFO scheduler.DAGScheduler: Missing parents: List()
2024-02-13 02:49:14,203 INFO scheduler.DAGScheduler: Submitting ShuffleMapStage 2 (MapPartitionsRDD[13] at save at DSET.scala:34), which has no missing parents

From these lines, we can infer the following:

A map stage job with ID 2 is created.
The final stage is ShuffleMapStage 2.
ShuffleMapStage 2 has no parents and no missing parents.
Finally, ShuffleMapStage 2 is submitted.
Based on this information, it appears that the shuffling is happening after the aggregation operation (group by and count), as the shuffle stage (ShuffleMapStage 2) is created after the

aggregation step. The aggregation is typically performed locally within each partition before shuffling the results across the network for further processing or storage.

Final conclusion: Other than RDD-G, all other programs do aggregation before shuffling. They do the aggregation locally on each partition of the RDD, thereby reducing the amount of data that needs to be shuffled across the network. This local aggregation happens within each partition and is done in memory.

4. [30 points] Show the pseudo-code for the 4 triangle-counting programs, including MAX-filter functionality. Since many Scala functions are similar to pseudo-code, you may copy-and-paste well-designed (good variable naming!) and well-commented Scala code fragments here.

a. **[10 points] RS-R**

```
val nodesCSV = sc.textFile(args(0))

  val filteredCSV = nodesCSV.filter(nodeLine => {
    val nodes = nodeLine.split(",")
    val follower = nodes(0).toLong
    val followed = nodes(1).toLong

    follower.toLong < MAX_FILTER && followed.toLong < MAX_FILTER
  })

  val followerFollowed = filteredCSV.map(nodeLine =>{
    val nodes = nodeLine.split(",")
    val follower = nodes(0).toLong
    val followed = nodes(1).toLong

    (follower,followed)
  })

  val followedFollower = followerFollowed.map { case (key, value) =>
    (value, key)
  }

  val traingleCounter = sc.longAccumulator("Triangle Counter")

  val path2 = joinOnKey(followedFollower,followerFollowed).map(_._2)

  val triangles = joinOnKey(path2,followedFollower).map(_._2)

  triangles.foreach {t=>if(t._1 == t._2) traingleCounter.add(1)}
```

```
println("Number of Triangles: " + traingleCounter.value / 3)

val triangleCount = traingleCounter.value / 3
sc.parallelize(Seq(triangleCount)).saveAsTextFile(args(1))
```

b. [5 points] RS-D

```
val edgesDF: DataFrame = spark.read
  .format("csv")
  .option("inferSchema", "true") // Infer schema from data types
  .load(args(0))
  .toDF("follower", "followee") // Specify column names

import org.apache.spark.sql.functions._
val filteredEdgesDF = edgesDF
.filter(col("follower") < MAX_FILTER && col("followee") < MAX_FILTER)

import spark.implicits._

val path2DF = filteredEdgesDF.as("E1")
            .join(filteredEdgesDF.as("E2"))
            .where($"E1.followee" === $"E2.follower" && $"E1.follower" =!=
$"E2.followee")
            .select($"E1.follower".alias("E1Follower"),
                  $"E2.follower".alias("E2Follower"),
                  $"E2.followee".alias("E2followee"))

val triangles = path2DF
          .join(filteredEdgesDF,
            path2DF("E2followee") === filteredEdgesDF("follower") &&
            path2DF("E1follower") === filteredEdgesDF("followee"))
            .count() / 3

// path2DF.explain(true)
println(s"Number of triangles: $triangles")

spark.sparkContext.parallelize(Seq(triangles.toString)).saveAsTextFile(args(1))
```

c. [10 points] Rep-R

```
val nodesCSV = sc.textFile(args(0))

val filteredCSV = nodesCSV.filter(nodeLine => {
  val nodes = nodeLine.split(",")
  val follower = nodes(0).toLong
  val followed = nodes(1).toLong
```

```
  follower.toLong < MAX_FILTER && followed.toLong < MAX_FILTER
})

val followerFollowed = filteredCSV.map(nodeLine =>{
  val nodes = nodeLine.split(",")
  val follower = nodes(0).toLong
  val followed = nodes(1).toLong

  (follower,followed)
})

val followedFollower = followerFollowed.map { case (key, value) =>
  (value, key)
}

val traingleCounter = sc.longAccumulator("Triangle Counter")

val broadCastfollowerFollowed = followerFollowed.collect()
                              .groupBy { case (follower, followed) => follower }

val broadCastedfollowerFollowed = sc.broadcast(broadCastfollowerFollowed)

// Access the broadcasted value
val broadcastedValue = broadCastedfollowerFollowed.value


val path2 = followedFollower.flatMap {
  case(kL, vL) => broadCastedfollowerFollowed.value.get(kL) match {
    case Some(vS) => vS.map(element => (vL, element))
    case None => List.empty // Return an empty list if no value is found for kL
  }
}


val triangles = followedFollower.join(path2)



triangles.foreach {case (key, (n1,(n2,n3))) => if(n1 == n3) traingleCounter.add(1)}

val triangleCount = traingleCounter.value / 3
sc.parallelize(Seq(triangleCount)).saveAsTextFile(args(1))
```

```
println("Number of Triangles: " + traingleCounter.value / 3)
```

d. [5 points] Rep-D

```
val MAX_FILTER = args(2).toLong

  // Read input CSV file into DataFrame
  val edgesDF: DataFrame = spark.read
    .format("csv")
    .option("inferSchema", "true") // Infer schema from data types
    .load(args(0))
    .toDF("follower", "followee") // Specify column names

  import spark.implicits._

  import org.apache.spark.sql.functions._
  val filteredEdgesDF = edgesDF
  .filter(col("follower") < MAX_FILTER && col("followee") < MAX_FILTER)

  // Broadcasting edgesDF
  import org.apache.spark.sql.functions.broadcast

  val bcfilteredEdgesDF = broadcast(filteredEdgesDF)

  val path2DF = filteredEdgesDF.as("E1")
            .join(bcfilteredEdgesDF.as("E2")) // Broadcasting E2 DataFrame
            .where($"E1.followee" === $"E2.follower" && $"E1.follower" =!=
$"E2.followee")
            .select($"E1.follower".alias("E1Follower"),
                  $"E2.follower".alias("E2Follower"),
                  $"E2.followee".alias("E2followee"))

  val triangles = path2DF
          .join(bcfilteredEdgesDF,
            path2DF("E2followee") === bcfilteredEdgesDF("follower") &&
            path2DF("E1follower") === bcfilteredEdgesDF("followee"))
            .count() / 3

  // path2DF.explain(true)
  println(s"Number of triangles: $triangles")

  spark.sparkContext.parallelize(Seq(triangles.toString)).saveAsTextFile(args(1))
```

5. [16 points] Show the link(s) to the source code for these programs in your Github Classroom repository.

RS-R:
https://github.com/CS6240/hw-3-AkshaySyal/blob/master/src/main/scala/wc/RSRDD.scala

RS-D: https://github.com/CS6240/hw-3-AkshaySyal/blob/master/src/main/scala/wc/RSD.scala

Rep-R:
https://github.com/CS6240/hw-3-AkshaySyal/blob/master/src/main/scala/wc/RepRDD.scala

Rep-D: https://github.com/CS6240/hw-3-AkshaySyal/blob/master/src/main/scala/wc/RepD.scala

6. [8 points] Run each triangle-counting program on EMR using 1 master and 4 worker nodes, using the same machine type and MAX setting as for the corresponding MapReduce program (RS-R and RS-D like RS-join in HW 2; Rep-R and Rep-D like Rep-join in HW 2). If a program runs faster than 15 min for the same settings as HW 2, simply report the faster time. If a program runs more than twice as long as the corresponding MapReduce version, you may choose a lower MAX setting. If the Spark version crashes with out-of-memory error for the same settings where the MapReduce program worked fine, explore Spark parameters to control memory size (container size, heap size), but do not use larger machine instances. If you still cannot resolve the memory issue, report the settings you tried and fix the problem by using a lower MAX. For each of the 4 programs (RS-R, RS-D, Rep-R, Rep-D), report machine type, MAX value, running time, and triangle count. For comparison, also report machine type, MAX, running time, and triangle count for the corresponding join versions from HW 2. The machine type must be the same. If you changed MAX, briefly explain why (see discussion above).

HW3	Machine Type	MAX	Running time	Triangle Count
RS-R	m3.xlarge	10,000	3min 19sec	520,296
RS-D	m3.xlarge	10,000	2 min 32 sec	520,296
Rep-R	m3.xlarge	10,000	7 min 9 sec	520,296
Rep-D	m3.xlarge	10,000	1 min 49 sec	520,296

7. [4 points] Show the link to the stderr file and the file that contains the triangle count for the run that produced the above running-time numbers for each of the 4 programs (RS-R, RS-D, Rep-R, Rep-D).

RS-R: output stderr
RS-D: output stderr
Rep-R: output stderr
Rep-D: output stderr

8. [8 points] Run the triangle-counting programs on the larger cluster with 1 master and 8 workers (or 7, if 8 caused quota issues on AWS Academy), following the same instructions (same configuration as for HW 2; change MAX only if necessary). For each of the 4 programs (RS-R, RS-D, Rep-R, Rep-D), report machine type, MAX, running time, and triangle count. For comparison, also report machine type, MAX, running time, and triangle count for the corresponding join versions from HW 2. The machine type must be the same. If you changed MAX, briefly explain why (see discussion above).

HW3	Machine Type	MAX	Running Time	Triangle Count
RS-R	m3.xlarge	10,000	2 min 42s	520,296
RS-D	m3.xlarge	10,000	1min 38s	520,296
Rep-R	m3.xlarge	10,000	3min 17s	520,296
Rep-D	m3.xlarge	10,000	1min 44s	520,296

9. [4 points] Show the link to the stderr file and the file that contains the triangle count for the run that produced the above running-time numbers for each of the 4 programs (RS-R, RS-D, Rep-R, Rep-D).

RS-R: output stderr
RS-D: output stderr
Rep-R: output stderr
Rep-D: output stderr