Class: 6240 - Large Scale Parallel Data Processing
HW: 3
Name: Sneh Gurdasani (NUID: 001399060)

Pseudo-Code for Twitter-follower:

RDD-G:

```
val title = textFile.map(line => line.split(",")(1)).map(word =>(word,1))
// Group by key
val counts = title.groupByKey().map(t=>t._1+","+t._2.sum)
```

RDD-R:

```
val title = textFile.map(line => line.split(",")(1)).map(word =>(word,1))
 // Reduce by key
 val reduce_by_key_out = title.reduceByKey(_ + _)
```

RDD-F:

```
val title = textFile.map(line => line.split(",")(1)).map(word =>(word,1))
// Fold by key
val fold_key_output = title.foldByKey(0)(_+_)
```

RDD-A:

```
val title = textFile.map(line => line.split(",")(1)).map(word =>(word,1))
// Aggregate by key
val aggregate_key_output = title.aggregateByKey(0)(_+_,_+_)
```

DSET:

```
val title = textFile.map(line => line.split(",")(1)).map(word =>(word,1))
val ss = SparkSession
  .builder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()
```

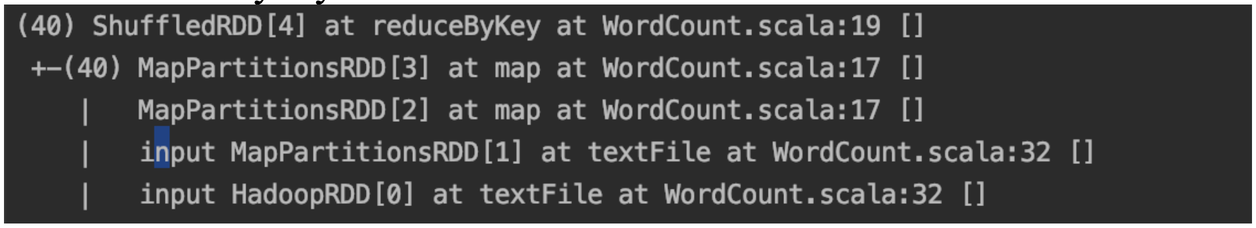
```
import ss.implicits._


val data = textFile.toDS()
val follow_pairs = data.map(line => (line.split(",")(1),1)).toDF("Followee","count").groupBy("Followee").count()
follow_pairs.explain(extended = true)
follow_pairs.write.csv(Outpath)
```

Programs which are performing Aggregation before Shuffle:
- RDD Reducebykey
- RDD Foldbykey
- RDD Aggregatebykey

Programs which are performing Aggregation after Shuffle:
- RDD Groupbykey
- DSET

toDebugString() and explan() outputs:

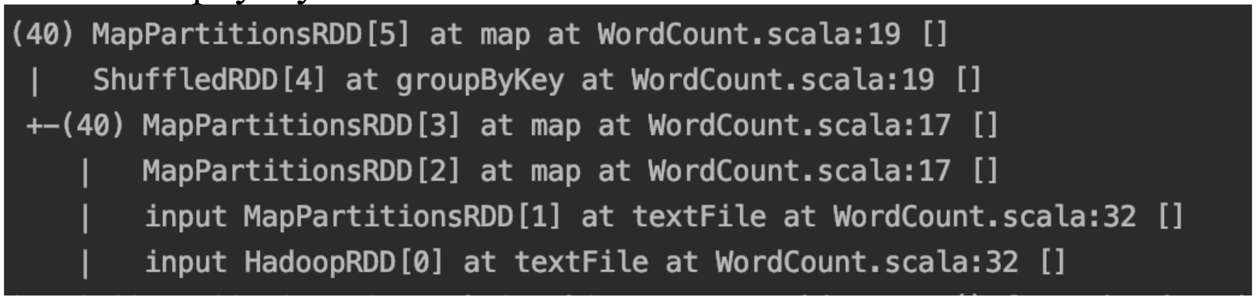
RDD Reducebykey:



```
(40) ShuffledRDD[4] at reduceByKey at WordCount.scala:19 []
 +-(40) MapPartitionsRDD[3] at map at WordCount.scala:17 []
    |   MapPartitionsRDD[2] at map at WordCount.scala:17 []
    |   input MapPartitionsRDD[1] at textFile at WordCount.scala:32 []
    |   input HadoopRDD[0] at textFile at WordCount.scala:32 []
```

"Update":5027920,"Value":5027920

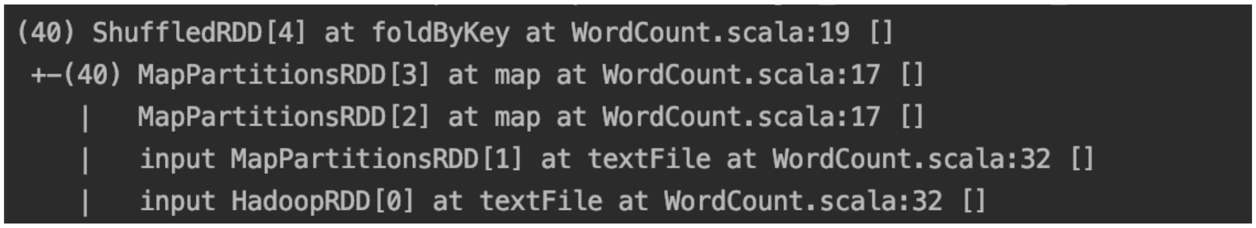
RDD Groupbykey:



```
(40) MapPartitionsRDD[5] at map at WordCount.scala:19 []
 |   ShuffledRDD[4] at groupByKey at WordCount.scala:19 []
 +-(40) MapPartitionsRDD[3] at map at WordCount.scala:17 []
    |   MapPartitionsRDD[2] at map at WordCount.scala:17 []
    |   input MapPartitionsRDD[1] at textFile at WordCount.scala:32 []
    |   input HadoopRDD[0] at textFile at WordCount.scala:32 []
```

"Update":26973021,"Value":26973021

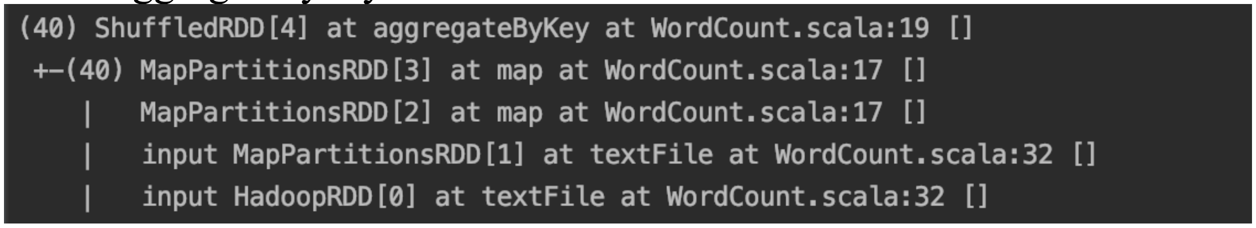
RDD Foldbykey:



```
(40) ShuffledRDD[4] at foldByKey at WordCount.scala:19 []
 +-(40) MapPartitionsRDD[3] at map at WordCount.scala:17 []
    |   MapPartitionsRDD[2] at map at WordCount.scala:17 []
    |   input MapPartitionsRDD[1] at textFile at WordCount.scala:32 []
    |   input HadoopRDD[0] at textFile at WordCount.scala:32 []
```

"Update":5209200,"Value":10239201

RDD Aggregatebykey:



```
(40) ShuffledRDD[4] at aggregateByKey at WordCount.scala:19 []
 +-(40) MapPartitionsRDD[3] at map at WordCount.scala:17 []
    |   MapPartitionsRDD[2] at map at WordCount.scala:17 []
    |   input MapPartitionsRDD[1] at textFile at WordCount.scala:32 []
    |   input HadoopRDD[0] at textFile at WordCount.scala:32 []
```

shuffle.write.bytesWritten","Update":360720,"Value":119675000

DSET:

```
== Physical Plan ==
*(2) HashAggregate(keys=[Followee#11], functions=[count(1)], output=[Followee#11,
count#18L])
+- Exchange hashpartitioning(Followee#11, 200)
   +- *(1) HashAggregate(keys=[Followee#11], functions=[partial_count(1)],
output=[Followee#11, count#23L])
      +- *(1) Project [_1#8 AS Followee#11]
         +- *(1) SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String,
StringType, fromString, assertnotnull(input[0, scala.Tuple2, true])._1, true, false) AS _1#8,
assertnotnull(input[0, scala.Tuple2, true])._2 AS _2#9]
            +- *(1) MapElements wc.WordCountMain$$$Lambda$1142/1179314953@e162a35,
obj#7: scala.Tuple2
               +- Scan input[obj#1]
```

```
shuffle.write.bytesWritten","Update":360720,"Value": 119675000
```

Programs which are performing Aggregation before Shuffle:
- RDD Reducebykey
- RDD Foldbykey
- RDD Aggregatebykey

Programs which are performing Aggregation after Shuffle:
- RDD Groupbykey
- DSET

Pseudo-Code for joins:

- I have implemented Max-filter in join program only.

Reduce side RDD Join:

```
val edges1 = textFile
```

```
                    .map(line => (line.split(",")(0).toInt,line.split(",")(1).toInt))
                    .filter(row_vals => row_vals._1 <= 10000 & row_vals._2 <= 10000)
val rev_edges = textFile
                              .map(line => (line.split(",")(1).toInt,line.split(",")(0).toInt))
                              .filter(row_vals => row_vals._1 <= 10000 & row_vals._2 <= 10000)

val native_edges = textFile
                              .map(line => (line.split(",")(0).toInt,line.split(",")(1).toInt))
                              .filter(row => row._1 <= 10000 & row._2 <= 10000)
                              .map(row1 => ((row1._1, row1._2),1))

val pathOf2 = edges1
                              .join(rev_edges)
                              .map(row => ((row._2._1, row._2._2),1))
val countOfTriangles = pathOf2
                                        .join(native_edges)
                                        .count()
Return countOfTriangles
```

Reduce Side DataSet Join:

```
val edgesDataSet = textFile
                              .map(line => (line.split(",")(0).toInt,line.split(",")(1).toInt))
                              .filter(row_vals => row_vals._1 <= 10000 & row_vals._2 <= 10000)
                              .toDS()

val pathOf2 = edgesDataSet.as("Edges1")
                    .join(edgesDataSet.as("Edges2"))
                    .where($"Edges1._2" === $"Edges2._1")
                    .select($"Edges1._1", $"Edges2._2")

val noOfTriangles = pathOf2.as("Edges3")
                              .join(edgesDataSet.as("Edges1"))
                              .where($"Edges3._1" === $"Edges1._2" && $"Edges3._2" ===
$"Edges1._1")
                              .count()

sc.parallelize(Seq("No of Triangles", noOfTriangles / 3)).saveAsTextFile(args(1))
```

Replicated Join RDD:

```
val edges1 = textFile.map(line => (line.split(",")(0).toInt, line.split(",")(1).toInt)).filter(row =>
row._1 <= 50000 & row._2 <= 50000)

val revEdges = textFile.map(line => (line.split(",")(1).toInt, line.split(",")(0).toInt)).filter(row =>
row._1 <= 50000 & row._2 <= 50000)
val smallRDDLocal = edges1.collectAsMap()
sc.broadcast(smallRDDLocal)

val path2 = revEdges.mapPartitions(
      iter => {
        iter.flatMap
        {
          case (k,v1 ) =>
            smallRDDLocal.get(k) match {
              case None => Seq.empty[((Int, Int), Int)]
              case Some(v2) =>  Seq(((v1, v2),1))
            }
        }
      }, preservesPartitioning = true)

val edges3 = sc.parallelize(smallRDDLocal.map(row1 => ((row1._1, row1._2),1)).toSeq)

val noOfTriangles = path2.join(edges3).groupByKey().count()

sc.parallelize(Seq("No of Triangles", noOfTriangles / 3)).saveAsTextFile(args(1))
```

Replicated Join DataSet:

```
val edgesDataSet = textFile.map(line => (line.split(",")(0).toInt, line.split(",")(1).toInt)).filter(row
=> row._1 <= 10000 & row._2 <= 10000).toDS()

    val broadCastDataSet = broadcast(edgesDataSet.as("broadcastEdges"))

val pathOf2 = edgesDataSet.as("Edges1").join(broadcast(broadCastDataSet))
      .where($"Edges1._2" === $"broadcastEdges._1")
      .select($"Edges1._1", $"broadcastEdges._2")

    val noOfTriangles = pathOf2.as("Edges3").join(broadcast(broadCastDataSet))
      .where($"Edges3._1" === $"broadcastEdges._2" && $"Edges3._2" ===
$"broadcastEdges._1")
      .count()
```

Configuration	Local Machine Result
RS-R, MAX = 10000	Running time: 4minutes 29s, Triangle count: 520311
RS-D, MAX = 10000	Running time:76s, Triangle count: 520311
Rep-R, MAX = 10000	Running time: 3 minutes, Triangle count: 0
Rep-DS, MAX = 10000	Running time: 49s, Triangle count: 520311

Output files for Combiner:

1. Reducebykey: ./Spark-Demo-Reducebykey/output

2. Groupbykey: ./Spark-Demo-Groupbykey/output

3.Aggregatebykey: ./Spark-Demo-Aggregatebykey/output

4. Foldbykey: ./Spark-Demo-Foldbykey/output

5.DSET:./Spark-Demo-Dataset/output

Output files for Joins:

1. Replicated Dataset: ./Spark-Demo-Rep-DS/output
2. Replicated RDD: ./Spark-Demo-Rep-RDD/output
3. Reduce Side Dataset: ./Spark-Demo-RS-DS/output
4. Reduce Side RDD: ./Spark-Demo-RS-RDD/output

Log links for Joins:
1. Replicated Dataset: ./Spark-Demo-Rep-DS/Logs
2. Replicated RDD: ./Spark-Demo-Rep-RDD/Logs
3. Reduce Side Dataset: ./Spark-Demo-RS-DS/Logs
4. Reduce Side RDD: ./Spark-Demo-RS-RDD/final_logs