Spark RDD Coding Problems with Solutions

Word count in Eclipse IDE:

-->reduceByKey: we **deal with only values based on keys**
For. mapWords.reduceByKey((x,y) =>x+y)
Ex. ("data",1)
("data",1)
Output: (“data”,2)
We deal with only values, so x+y= 1+1 (summing only the values)

```scala
import org.apache.log4j.Logger
import org.apache.spark.SparkContext
import org.apache.log4j.Level

object word_count extends App {

// it shows us logs of only Error and not process running logs
Logger.getLogger("org").setLevel(Level.ERROR)

//"sparkcontext" is entry pt to spark cluster. local[*] => we are on local machince and all cores can be utilized by spark cluster, "word_count" => appl name
  val sc=new SparkContext("local[*]","word_count")

  //we got the file from local, DAG created in backend as it is only transformation
  val input=sc.textFile("Downloads/search.txt")

  //flatmap => considers each line as input, splits each word in a line seperately
  val flatWords=input.flatMap(x=>x.split(" "))

  //map=> one-to-one mapping, for n inputs we have n outputs. creates tuple in form of (eachElement,1)
  val mapWords=flatWords.map(x=>(x,1))

  //sorts the words and performs the count for 2 similar elements at a time
  val reduceWords=mapWords.reduceByKey((x,y)=>x+y)

  //collects the output and prints each output. It is an action
  val finalCount=reduceWords.collect.foreach(println)

  //let's spark appl keep in running mode so we can see its UI in "localhost:4040"
  scala.io.StdIn.readLine()
}
```

Use cases:

1. When we have to convert all the words into lower case
2. When we need to sort out keywords with high frequency

```
//flatmap => considers each line as input, splits each word in a line seperately
val flatWords=input.flatMap(x=>x.split(" "))

  //convert words into lowercase
val toLower=flatWords.map(x=>x.toLowerCase())

//map=> one-to-one mapping, for n inputs we have n outputs. creates tuple in form of (eachElement,1)
val mapWords=flatWords.map(x=>(x,1))

//sorts the words and performs the count for 2 similar elements at a time
val reduceWords=mapWords.reduceByKey((x,y)=>x+y)

//to sort the values based on value(frequency) ie x._2 element which is a tuple
//we have only "sortByKey" not "sortByValue". so sorting can be done based on key else we have to reverse the order ie map(x=>x._2,x._1) key becomes value and value becomes key
val sortWords=reduceWords.sortBy(x=>x._2)
val sortCount=sortWords.collect

//print the results in a good format
for(result<-sortCount)
{
  val word=result._1
  val count=result._2
  println(s"$word,$count")
}
```

Customer Orders Program:

-->we need to find top customers who made highest purchase

Based on the file, choose the fields that are required to give the desired result

```
//Program Context
//we have file(cust_id, order_id, purchase amt)
//we have to find top 10 customers who made highest purchase
//we will be ignoring order_id as it not very significant

object spark_orders extends App {

    // it shows us logs of only Error and not process running logs
Logger.getLogger("org").setLevel(Level.ERROR)

val sc=new SparkContext("local[*]","orders")

val newFile=sc.textFile("Downloads/Customerorders.csv")

//we require 1st and 3rd element, and accessing it as an array element(each line split into array of elements)

val mappedOutput=newFile.map(x=>(x.split(" ")(0),x.split(" ")(2).toFloat ) )

//we then sum all the amts for a cust_id
val reduceOutput=mappedOutput.reduceByKey((x,y)=>x+y)

//we sort it based on "amt" field
val sortOutput=reduceOutput.sortBy(x=>x._2)

//display the results
val result=sortOutput.collect.foreach(println)

scala.io.StdIn.readLine()

}
```

Customer Movie Ratings:

-->we need to find count of movies which got rating 5, 4,3,2,1

```
object spark_movie  {

 //context of the program
  //cust_id, movie_id, movie_rating,time of watch
  //we need to find out total no. of movies (count) which were rated 5,4,3,2,1

  def main(args:Array[String])
  {

  val sc=new SparkContext("local[*]","movie")

  val newFile=sc.textFile("Downloads/moviedata.data")

  //we require only "movie rating" field
  val mapMovie=newFile.map(x=>x.split(" ")(2))

  //converting it into a form that can be easily reduced ex. (5,1) (5,1), (4,1),(4,1),(4,1)
  val tupleMovie=mapMovie.map(x=>(x,1))

  //reduce the results and find the final count
  val reduceMovie=tupleMovie.reduceByKey((x,y)=>(x+y))

  val finalResult=reduceMovie.collect.foreach(println)

  println(finalResult)
```

```
//*********METHOD 2**********************
/*
 val mapMovie=newFile.map(x=>x.split(" ")(2))

 //from mapper itself we can get output rather than going to reducer
  val finalCount=mapMovie.countByValue

 finalCount.collect.foreach(println)
 */
```

Average LinedIn Connections based on Age:

--> we need to find avg no. of connections based on age

-->Ex. (24,560) (25,100)

```
  //problem context
  //row_number,person_name,age,linkedInConnections
  //we need to find avg of connections based on age
  //(age,avgConn) => (25,360)



  def arrayMap(line:String)=
  {

    val age=line.split(",")(2).toInt
    val conn=line.split(",")(3).toInt
    (age,conn)
  }
      // it shows us logs of only Error and not process running logs
Logger.getLogger("org").setLevel(Level.ERROR)

val sc=new SparkContext("local[*]","orders")

val newFile=sc.textFile("Downloads/Customerorders.csv")

//input
//(1,"vaishu",24,650)
//(2,"yashe",27,210)

//output
//(24,650)
//(27,210)
```

```scala
//can be written as below as well
//val mapInput=newFile.map(x=>(x.split(",")(2).toInt),x.split(",")(3).toInt)
val mapInput=newFile.map(arrayMap)

//input
//(24,650)

//output
//(24,650,1)

//appending each tuple element with "1" to find the count finally
val mapOutput=mapInput.map(x=>(x._1,(x._2,1)))

//input
//(24,650,1)
//(24,250,1)

//output
//(24,900,2)

//sum(conn) + sum (people in that age)
//x._1= conn x._2=count
val avgFriend=mapOutput.reduceByKey((x,y)=>((x._1+y._1,x._2+y._2)))

//input
//(24,850,2)

//output
//(24,450)  ie (24,900/2)
//conn= x._2._1, count=x._2._2
val finalResult=avgFriend.map(x=>(x._1,x._2._1/x._2._2) )


//input
//(24,850,2)

//output
//(24,450)  ie (24,900/2)
//conn= x._2._1, count=x._2._2
val finalResult=avgFriend.map(x=>(x._1,x._2._1/x._2._2) )

//print the result
val resultFriend=finalResult.collect.foreach(println)
```

Problem based on age

File contains name,age,city. You need to compare the age and if age > 18 then add a new column with value "Y"

If age <18 add a new column with value "N"

-->Got to know we can perform the entire operation inside the map function itself

```
object spark_assignment_age extends App {

   // it shows us logs of only Error and not process running logs
Logger.getLogger("org").setLevel(Level.ERROR)

val sc=new SparkContext("local[*]","orders")

val newFile=sc.textFile("C:/Users/Chakka Yashwanth/Downloads/age.dataset1")

val ageValue=newFile.map(x=>{

val fields=x.split(",")

if(fields(1).toInt >= 18)
{
  (fields(0),fields(1),fields(2),"Y")
}

else
{
  (fields(0),fields(1),fields(2),"N")
}
}
)

ageValue.collect.foreach(println)
```

Find minimum temperature

File contains stationId, TimeOfReading,
ReadingType,TemperatureRecorded,
Find the min temp of each station id

```
val newFile=sc.textFile("C:/Users/Chakka Yashwanth/Downloads/tempdata.csv")
//input
//station_id,time,temp_type,temp,....

//output
//Array(station_id  temp_temp    temp)

val mapFile=newFile.map(x=>(x.split(",")(0),x.split(",")(2),x.split(",")(3).toFloat  ))

//input
//(TEO1 TMIN -56)
//(TEO1 TMAX -6)

//OUTPUT
//(TEO1 TMIN -56)
val filterMap=mapFile.filter(x=>(x._2 == "TMIN"))  //Filter out records which belong to "TMIN"

//input
//(TEO1 TMIN -56)

//output
//(TEO1 -56)

val finalMap=filterMap.map(x=>(x._1,x._3.toFloat))

//find the minimum among all the values we have for each station id
val reduceFile=finalMap.reduceByKey((x,y)=> min(x,y))

//print the final results
val finalResult=reduceFile.collect.foreach(println)
```

To find cost of top key words

-->we have a list of top keywords through big data topics were searched for.

-->we have to find the cost of each keyword which we spend on google ad campaign

-->we deal with price(field 10) and keyword(field 0)

```
  Logger.getLogger("org").setLevel(Level.ERROR)

val sc=new SparkContext("local[*]","orders")

val newFile=sc.textFile("C:/Users/Chakka Yashwanth/Downloads/bigdata.csv")

//output
//Array((big data course,25),(big data hadoop,100))
val initialMap=newFile.map(x=>(x.split(",")(10).toFloat,x.split(",")(0).toString) )

//input
//(25,big data course)

//output
//(25,big)
//(25,data)
//(25,course)

//order to apply(float/int, string)
val flatValues=initialMap.flatMapValues(x=>x.split(" "))

//input (25,big) //output (big,25)
// so that we can use reduceByKey
val mapFinal=flatValues.map(x=>(x._2,x._1))

//input
//(big,25)
//(big,35)

//output
//(big,70)

//** to use "reduceByKey" we shd have key values on "LEFT"

val finalResult=mapFinal.reduceByKey((x,y)=>x+y)
```

```scala
val finalResult=mapFinal.reduceByKey((x,y)=>x+y)
//input
//(big,1200)
//data(1800)
//(hadoop,1250)

//output
//(data,1800)
//(hadoop,1250)
//(big,1200)

//x._2 is "price" and false="descending order"
val sortOrder=finalResult.sortBy(x=>x._2,false)

sortOrder.collect.foreach(println)
```

****Questions to ask before solving a problem?**

1. Does this problem need a reducer or mapper only can provide the final output?
2. Does each input line has to be split into an array
3. Which field do we actually require to fulfill our request
4. What kind of transformation should we use? (ex. map, flatMap)
5. Examine the data columns carefully based on which you get data insights that can be used in your problem
6. Remove the duplicates in data if it hampers your results