SPARK RDD Transformations & Actions:

Queries and explanations:

map() ==> give one-to one mapping ie if we have **1 i/p we get only 1 output**
1.1 **To convert input into an array**
-->map(x=>x.split(",")) ==>it will split the input and convert into an array
//input
//vaish,big data,reading,23

//output
//Array(vaish big data reading 23)

1.2 To access particular elements of the rdd tuple
X.map(x=>x._1,x._3)

//input
//("vaish", "passion","big data")

//output
//("vaish","big data")

1.3 To reverse elements of the rdd tuple
X.map(x=>x._2,x._1)

//input
//("vaish","big data")

//output
//("big data","vaish)

1.4 To access elements of sub tuple rdd
X.map(x=>x._1,(x._2,**x._2._1)**)

1.5 To directly work on "map" values
X.**mapValues**(x => (x._1/x._2))
//input
//("vaish",100,200)
//("vaish",300,300)

//output
//("vaish",0.5)
//("vaish",1)

1.6 To add values to rdd tuple
X.map(x=>x,1)

("yashe") ("vaish") ==> ("yashe",1) ("vaish",1)

1.7 To perform map operations on partition level
X.mapPartition(x=>(x,1))

1.8 To perform map operations on partition level and track them
X.mapPartitionwithIndex(x=>(x,1))
Here we can track the index of the partition whose mapping operations are ongoing

2.**flatMap()** : performs one-to-many mapping ie **one i/p gives many outputs**
2.1 x.**flatMap**(x=>x.split(","))
2.2 x.filterMap(x=>x.split(","), 1) ==>to add values to rdd tuple

("vaishu","yashe","anju") ==> ("vaishu") ("yashe") ("vaishu")
("vaishu","yashe","anju") ==> ("vaishu",1) ("yashe",1) ("vaishu",1)

2.1 To flatten the map based on values
2.1.1 rdd.**flatMapValue**(x=>x.split(" "))

(big data course, 25) => (big, 25) (data,25) (course,25)

3.**filter()** : To filter out tuples based on condition
X.filter(x=>x._2 == "data")

(1, data) (2,big) (3,data) ==> (1,data) (3,data)

4.**reduceByKey**() : To perform aggregations on a set of values based on "key"
X.reduceByKey((x,y)=>x+y)
X.reduceByKey(x,y=>(x._1+y._1),(x._2,y._2))

(data,1) (big,1) (data,1) =>(data,2) (big,1)

(data,100,200) (big,1000,500) (data,500,300) =>(data,600,500) (big,1000,500)

5.**To sort the data**
5.1 x.**sortBy**(x=>x._2) //To sort data based on second field in tuple. It sorts in **ascending order**

5.2 x.**sortBy**(x=>(x._2),false) //It sorts the data in descending order

5.3 First transpose the elements to make "value" to be "key" rdd.map(x=>x._2,x._1) . Now u can use below function
x.**sortByKey**(x=>x._1) //It is a transformation (**it is even shown as a action due to some inner implementation)

7.To join the tables
Val joinRdd=newRdd.**join**(firstRdd)
newRdd==>**chapter_id**,user_id
firstRdd==>**chapter_id**,course_id

-->when u r joining 2 tables/files, make sure the first field is "**join field**"
-->resultant rdd will be tuple of **(chapter_id, (user_id,course_id))**

8. To get count of values from mapper itself
Rdd2=rdd1.**countByValue**()
//input [5],[5],[5],[3],[5],[5],[3],[4]

//output [5,5], [4,1] , [3,2]

9. **To increase/decrease no. partitions**

Rdd.repartition(10) //to increase no. of partitions
Rdd.coalesce(3) //to decrease no. of partitions

10. **To group the output based on key**

Rdd.groupByKey(x,y=>x+y) //u will learn more on this in next chapters

11. **To perform union/intersection of data sets**

Rdd.union(dataset)
Rdd.intersection(dataset)

12. **To get distinct values from a rdd**

Rdd.distinct()

ACTIONS:

1. To return all the output values from a transformation

Rdd.collect.foreach(println) // collect output and print each line

2. To return the first row from output

Rdd.first() //return first output row

3. To return n output rows

Rdd.take(n)

4. To return the no. of elements

Rdd.count()

5. To reduce the input into single line output

Rdd.reduce()

6. To save output into a file

6.1 x.**saveAsTextFile**("<output path>")
6.2 rdd.saveAsSequenceFile("path")
6.3 rdd.saveAsObjectFile("path")

7. To show first 20 records of a output

Rdd.show()

How to remove header from a file using Rdd ?

Val fileRdd=spark.sc.textFile("path")
Val firstElement=fileRdd.first() // header is returned
Val filterRdd=firstElement.filter(x=>!x.contains(firstElement)) //filter out the header
Val printResult=filterRdd.collect.foreach(prinltn) //print all elements after header filtered out

Repartition(), coalesce(), groupByKey(), distinct(), union(), intersection()
Count(), first(), take(),countByKey(), foreach(), saveAsSequeceFile(), saveAsObjectFile()

More on spark RDD:
https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

KEYWORDS:

Spark : In-memory processing engine
Why spark is fast: Due to less I/O disc reads and writes
RDD: It is a data structure to store data in spark
When RDD fails: Using lineage graph we track which RDD failed and reprocess it
Why RDD immutable : As it has to be recovered after its failure and to track which RDD failed
Operations in spark: Transformation and Action
Transformation: Change data from one form to another, are lazy.
Action: Operations which processes the tranformations, not lazy. creates DAG to remember sequence of steps.
Port number: localhost://4040 → spark UI

**Note:
1 hdfs block = 1 rdd partition = 128mb
1 hdfs block in local=1 rdd partition in local spark cluster= 32mb
1 rdd ~ can have n partitions in it
1 cluster = 1 machine
N cores = N blocks can run in parallel in each cluster/machine
N stages = N - 1 wide transformations
N tasks in each stage= N partitions in each stage for that rdd/data frame