databricks Scala Notebook 2023-08-21 11:07:59

(https://databricks.com)

```
// comment in scala


var a  = Array(List(1,2,3), List(1,2,3), List(1,2,3))
var b  = Array(List(4,5,6), List(4,5,6), List(4,5,6))
var c  = Array(Array(0,0,0), Array(0,0,0), Array(0,0,0))
var sum = 0

for (i<-0 to 2){
  for (j<-0 to 2){
    sum=0
    for(k<-0 to 2){
      sum = sum + (a(i)(k) * b(k)(j))
    }
    c(i)(j)=sum
  }
}
println(c)
```

```
[[I@696395e5
a: Array[List[Int]] = Array(List(1, 2, 3), List(1, 2, 3), List(1, 2, 3))
b: Array[List[Int]] = Array(List(4, 5, 6), List(4, 5, 6), List(4, 5, 6))
c: Array[Array[Int]] = Array(Array(24, 30, 36), Array(24, 30, 36), Array(24, 30, 36))
sum: Int = 36
```

```
a(1)(1)
```

```
res2: Int = 2
```

```
var a =Array(Array(1,2,3), List(1,2,3), List(1,2,3))
```

```
a: Array[java.io.Serializable] = Array(Array(1, 2, 3), List(1, 2, 3), List(1, 2, 3))
```

```
a(1)
```

```
res3: java.io.Serializable = List(1, 2, 3)
```

```
a(0)
```

```
res4: java.io.Serializable = Array(1, 2, 3)
```

Transportation function examples

```
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.SparkSession
```

```
object RDDParallelize{
  def main(args: Array[String]): Unit = {
    val spark:SparkSession = SparkSession.builder().master("local[1]").appName("SparkByExamples.com").getOrCreate()
    val rdd:RDD[Int]= spark.sparkContext.parallelize(List(1,2,3,4,5))
    val rddCollect:Array[Int]= rdd.collect()
    println("Number of partitions:" +rdd.getNumPartitions)
    println("Action: First element:" +rdd.first)
    println("Action: RDD converted to Array[Int]: ")
    rddCollect.foreach(println)
  }
}
```

```
defined object RDDParallelize
```

```
val rdda = sc.parallelize(List(1,2,3,4,5))
    val rddb = rdda.collect
    println("Number of Partition:" +rdda.getNumPartitions)
    println("Action: First element:" + rdda.first())
    rdda.foreach(println)
```

```
Number of Partition:8
Action: First element:1
rdda: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at command-3160921003574745:1
rddb: Array[Int] = Array(1, 2, 3, 4, 5)
```

```
val  rdda = sc.parallelize(List("mumbai", "Delhi", "Chennai", "Kolkatta"))
```

```
rdda: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[1] at parallelize at command-3160921003574746:1
```

```
var rddb = sc.parallelize(Array(1,2,3,4,5,6,7,8,9,10))
```

```
rddb: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[2] at parallelize at command-3160921003574747:1
```

```
var rddc = sc.parallelize(Seq.empty[String])
```

```
rddc: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[3] at parallelize at command-3160921003574748:1
```

```
rdda.collect
```

```
res8: Array[String] = Array(mumbai, Delhi, Chennai, Kolkatta)
```

```
rddb.collect
```

```
res9: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
rddc.collect
// Collect is action
```

```
res11: Array[String] = Array()
```

```
  val b = rdda.map(x => (x,1))
  //  map, filter is trnasformation
```

```
b: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[5] at map at command-3160921003574752:1
```

```
  b.collect
```

```
res12: Array[(String, Int)] = Array((mumbai,1), (Delhi,1), (Chennai,1), (Kolkatta,1))
```

```
  val b = rdda.map((_,1))
```

```
b: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[6] at map at command-3160921003574754:1
```

```
  b.collect
```

```
res13: Array[(String, Int)] = Array((mumbai,1), (Delhi,1), (Chennai,1), (Kolkatta,1))
```

```
  val b = rdda.map(x =>(x,x.length))
```

```
b: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[7] at map at command-3160921003574756:1
```

```
  b.collect
```

```
res14: Array[(String, Int)] = Array((mumbai,6), (Delhi,5), (Chennai,7), (Kolkatta,8))
```

```
  // Word Length

  val a = sc.parallelize(List(1,2,3,4,5)).map(x=>List(x,x,x)).collect
```

```
a: Array[List[Int]] = Array(List(1, 1, 1), List(2, 2, 2), List(3, 3, 3), List(4, 4, 4), List(5, 5, 5))
```

```
  val a = sc.parallelize(List(1,2,3,4,5)).flatMap(x=>List(x,x,x)).collect
```

```
a: Array[Int] = Array(1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5)
```

```
  // What is difference between the map and flatmap
```

```
  // applying filter on rdda city data
  val rdda = sc.parallelize(List("Mumbai", "Mumbai", "Delhi", "Chennai", "Kolakatta")).filter(_.equals("Mumbai")).count
```

```
rdda: Long = 2
```

```
  // applying filter which contain a in city data
  val rdda = sc.parallelize(List("Mumbai", "Delhi", "Chennai", "Kolakatta")).filter(_.contains("e")).collect
```

```
rdda: Array[String] = Array(Delhi, Chennai)
```

```
// Creating an rdda with city,Count
val a = sc.parallelize(List(("Mumbai",4000),( "Delhi", 2000), ("Chennai",1000),("Kolakatta", 7000)))
```

```
a: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[18] at parallelize at command-3160921003574763:2
```

```
// Perform filter operation where value euals 4000
// _._1 is for key _._2 is for value
val a = sc.parallelize(List(("Mumbai",4000),( "Delhi", 2000), ("Chennai",1000),("Kolakatta",
7000))).filter(_._2.equals(4000)).collect
```

```
a: Array[(String, Int)] = Array((Mumbai,4000))
```

```
// Perform filter operation where values grather than 3000
val a = sc.parallelize(List(("Mumbai",4000),( "Delhi", 2000), ("Chennai",1000),("Kolakatta", 7000))).filter(_._2 > 3000).collect

// _._1 is for key _._2 is for value
```

```
a: Array[(String, Int)] = Array((Mumbai,4000), (Kolakatta,7000))
```

```
// Perform filter which start with C

val a = sc.parallelize(List(("Mumbai",4000),( "Delhi", 2000), ("Chennai",1000),("Kolakatta", 7000))).filter(a=>
a._1.startsWith("C")).collect

// Perform filter by range between 3000, 9000

val b = sc.parallelize(List((4000,"Mumbai"),(2000, "Delhi"), (1000, "Chennai"),(7000, "Kolakatta"))).filterByRange(3000,
9000).collect
```

```
a: Array[(String, Int)] = Array((Chennai,1000))
b: Array[(Int, String)] = Array((4000,Mumbai), (7000,Kolakatta))
```

```
// sample (flase/true, fraction, seed)
// false - can not have repeated values
// true- will have repeated values
//  seed - result will be same if the seed is kept same
```

```
val a = sc.parallelize(1 to 100)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[33] at parallelize at command-3160921003574768:1
```

```
a.sample(false, .2,5).collect
```

```
res17: Array[Int] = Array(2, 3, 7, 8, 13, 22, 25, 39, 43, 56, 59, 61, 66, 71, 73, 79, 83, 92)
```

```
a.sample(true, .2,5).collect
```

```
res19: Array[Int] = Array(2, 3, 9, 10, 13, 14, 19, 20, 24, 40, 43, 46, 56, 61, 62, 67, 80, 81, 82, 82, 92)
```

```
a.sample(false, .2).collect
```

```
res20: Array[Int] = Array(2, 18, 23, 32, 38, 45, 46, 47, 57, 60, 64, 73, 74, 81, 83, 86, 96)
```

```
a.sample(false,1,5).collect
```

```
res21: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100)
```

```
val a = sc.parallelize(List(1,2,1,1,1,2))
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[38] at parallelize at command-3160921003574773:1
```

```
a.sample(true, .4,5).collect
```

```
res22: Array[Int] = Array(1)
```

```
val a = sc.parallelize(1 to 7)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[40] at parallelize at command-3160921003574775:1
```

```
val b = sc.parallelize(5 to 10)
```

```
b: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[41] at parallelize at command-3160921003574776:1
```

```
a.union(b).collect
```

```
res23: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 5, 6, 7, 8, 9, 10)
```

```
a.intersection(b).collect
```

```
res24: Array[Int] = Array(5, 6, 7)
```

```
a.union(b).distinct.collect
```

```
res25: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
val b = sc.parallelize(1 to 9,3)
```

```
b: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[53] at parallelize at command-3160921003574780:1
```

```
b.mapPartitions(x=>List(x.next).iterator).collect
```

```
res28: Array[Int] = Array(1, 4, 7)
```

```
val a = sc.parallelize(1 to 9,4)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[57] at parallelize at command-3160921003574782:1
```

```
a.mapPartitions(x=>List(x.next).iterator).collect
```

```
res29: Array[Int] = Array(1, 3, 5, 7)
```

```
def practfunct(index: Int, iter:Iterator[(Int)]): Iterator[String] = {
  iter.toList.map(x => "[index :" +index + ", val : " +x +"]").iterator
}
```

```
practfunct: (index: Int, iter: Iterator[Int])Iterator[String]
```

```
val a  = sc.parallelize(List(1,2,3,4,5,6),2)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[59] at parallelize at command-3160921003574785:1
```

```
a.collect
```

```
res30: Array[Int] = Array(1, 2, 3, 4, 5, 6)
```

```
a.mapPartitionsWithIndex(practfunct).collect
```

```
res31: Array[String] = Array([index :0, val : 1], [index :0, val : 2], [index :0, val : 3], [index :1, val : 4], [index :1, val :
5], [index :1, val : 6])
```

```
val a = sc.parallelize(List(1,2,3,4,5,6),3)
```

```
a: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[61] at parallelize at command-3160921003574788:1
```

```
a.mapPartitionsWithIndex(practfunct).collect
```

```
res32: Array[String] = Array([index :0, val : 1], [index :0, val : 2], [index :1, val : 3], [index :1, val : 4], [index :2, val :
5], [index :2, val : 6])
```

```
val boradcastVar = sc.broadcast(Array(1,2,3))
boradcastVar.value
```

```
boradcastVar: org.apache.spark.broadcast.Broadcast[Array[Int]] = Broadcast(41)
res33: Array[Int] = Array(1, 2, 3)

a: Double = 2500.0
res34: Double = 2500.0
```