Free Questions for CCA175 by braindumpscollection

Shared by Wood on 24-05-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

Problem Scenario 51 : You have been given below code snippet.

val a = sc.parallelize(List(1, 2,1, 3), 1)

val b = a.map((_, "b"))

val c = a.map((_, "c"))

Operation_xyz

Write a correct code snippet for Operationxyz which will produce below output.

Output:

Array[(lnt, (lterable[String], lterable[String]))] = Array(

(2,(ArrayBuffer(b),ArrayBuffer(c))),

(3,(ArrayBuffer(b),ArrayBuffer(c))),

(1,(ArrayBuffer(b, b),ArrayBuffer(c, c)))

)

Options:

A- Solution :

b.cogroup(c).collect

cogroup [Pair], groupWith [Pair]

A very powerful set of functions that allow grouping up to 3 key-value RDDs together using their keys.

Another example

val x = sc.parallelize(List((1, 'apple'), (2, 'banana'), (2, 'orange'), (6, 'kiwi')), 2)

val y = sc.parallelize(List((5, 'computer'), (1, 'laptop'), (1, 'desktop'), (4, 'iPad')), 2)

x.cogroup(y).collect

Array[(Int, (Iterable[String], Iterable[String]))] = Array(

(4,(ArrayBuffer(kiwi),ArrayBuffer(iPad))),

(2,(ArrayBuffer(banana),ArrayBuffer())),

(1 ,(ArrayBuffer(apple),ArrayBuffer(laptop, desktop))),

(5,{ArrayBuffer(),ArrayBuffer(computer))))

B- Solution :

b.cogroup(c).collect

cogroup [Pair], groupWith [Pair]

A very powerful set of functions that allow grouping up to 3 key-value RDDs together using their keys.

Another example

val x = sc.parallelize(List((1, 'apple'), (2, 'banana'), (3, 'orange'), (4, 'kiwi')), 2)

val y = sc.parallelize(List((5, 'computer'), (1, 'laptop'), (1, 'desktop'), (4, 'iPad')), 2)

x.cogroup(y).collect

Array[(Int, (Iterable[String], Iterable[String]))] = Array(

(4,(ArrayBuffer(kiwi),ArrayBuffer(iPad))),

(2,(ArrayBuffer(banana),ArrayBuffer())),
(3,(ArrayBuffer(orange),ArrayBuffer())),
(1 ,(ArrayBuffer(apple),ArrayBuffer(laptop, desktop))),
(5,{ArrayBuffer(),ArrayBuffer(computer))))

Answer:

B

Question 2

Question Type: **MultipleChoice**

Problem Scenario 50 : You have been given below code snippet (calculating an average score}, with intermediate output.

type ScoreCollector = (Int, Double)

type PersonScores = (String, (Int, Double))

val initialScores = Array(("Fred", 88.0), ("Fred", 95.0), ("Fred", 91.0), ("Wilma", 93.0), ("Wilma", 95.0), ("Wilma", 98.0))

val wilmaAndFredScores = sc.parallelize(initialScores).cache()

val scores = wilmaAndFredScores.combineByKey(createScoreCombiner, scoreCombiner, scoreMerger)

val averagingFunction = (personScore: PersonScores) => { val (name, (numberScores, totalScore)) = personScore (name, totalScore / numberScores)

}

val averageScores = scores.collectAsMap().map(averagingFunction)

Expected output: averageScores: scala.collection.Map[String,Double] = Map(Fred -> 91.33333333333333, Wilma -> 95.33333333333333)

Define all three required function , which are input for combineByKey method, e.g. (createScoreCombiner, scoreCombiner, scoreMerger).And help us producing required results.

Options:

A- Solution :
val createScoreCombiner = (score: Double) => (1, score)
val scoreCombiner = (collector: ScoreCollector, score: Double) => {
val (numberScores. totalScore) = collector (numberScores + 1, totalScore + score)
}
val scoreMerger= (collector-!: ScoreCollector, collector2: ScoreCollector) => { val (numScoresl. totalScorel) = collector! val (numScores2, tota!Score2) = collector (numScoresl + numScores2, totalScorel + totalScore2)
}

B- Solution :
val createScoreCombiner = (score: Double) => (1, score)
val scoreCombiner = (collector: ScoreCollector, score: Double) => {

}
val scoreMerger= (collector-!: ScoreCollector, collector2: ScoreCollector) => { val (numScoresl. totalScorel) = collector! val (numScores2, tota!Score2) = collector (numScoresl + numScores2, totalScorel + totalScore2)
}

Answer:

A

Question 3

Question Type: **MultipleChoice**

Problem Scenario 49 : You have been given below code snippet (do a sum of values by key}, with intermediate output.

val keysWithValuesList = Array("foo=A", "foo=A", "foo=A", "foo=A", "foo=B", "bar=C", "bar=D", "bar=D")

val data = sc.parallelize(keysWithValuesl_ist}

//Create key value pairs

val kv = data.map(_.split("=")).map(v => (v(0), v(l))).cache()

val initialCount = 0;

val countByKey = kv.aggregateByKey(initialCount)(addToCounts, sumPartitionCounts)

Now define two functions (addToCounts, sumPartitionCounts) such, which will produce following results.

Output 1

countByKey.collect

res3: Array[(String, Int)] = Array((foo,5), (bar,3))

import scala.collection._

val initialSet = scala.collection.mutable.HashSet.empty[String]

val uniqueByKey = kv.aggregateByKey(initialSet)(addToSet, mergePartitionSets)

Now define two functions (addToSet, mergePartitionSets) such, which will produce following results.

Output 2:

uniqueByKey.collect

res4: Array[(String, scala.collection.mutable.HashSet[String])] = Array((foo,Set(B, A}}, (bar,Set(C, D}}}

Options:

A- Solution :

val addToCounts = (n: Int, v: String) => n + 1

val sumPartitionCounts = (p1: Int, p2: Int} => p1 + p2
val addToSet = (s: mutable.HashSet[String], v: String) => s += v
val mergePartitionSets = (p1: mutable.HashSet[String], p2: mutable.HashSet[String]) => p1 ++= p2

B- Solution :
val addToCounts = (n: Int, v: String) => n + 1
val mergePartitionSets = (p1: mutable.HashSet[String], p2: mutable.HashSet[String]) => p1 ++= p2

Answer:

A

Question 4

Question Type: **MultipleChoice**

Problem Scenario 48 : You have been given below Python code snippet, with intermediate output.

We want to take a list of records about people and then we want to sum up their ages and count them.

So for this example the type in the RDD will be a Dictionary in the format of {name: NAME, age:AGE, gender:GENDER}.

The result type will be a tuple that looks like so (Sum of Ages, Count)

people = []

people.append({'name':'Amit', 'age':45,'gender':'M'})

people.append({'name':'Ganga', 'age':43,'gender':'F'})

people.append({'name':'John', 'age':28,'gender':'M'})

people.append({'name':'Lolita', 'age':33,'gender':'F'})

people.append({'name':'Dont Know', 'age':18,'gender':'T'})

peopleRdd=sc.parallelize(people) //Create an RDD

peopleRdd.aggregate((0,0), seqOp, combOp) //Output of above line : 167, 5)

Now define two operation seqOp and combOp , such that

seqOp : Sum the age of all people as well count them, in each partition. combOp : Combine results from all partitions.

Options:

A- Solution :

seqOp = (lambda x,y: (x[6] + y['age'],x[2] + 1))

combOp = (lambda x,y: (x[0] + y[0], x[0] + y[1]))

B- Solution :

seqOp = (lambda x,y: (x[0] + y['age'],x[1] + 1))

combOp = (lambda x,y: (x[0] + y[0], x[1] + y[1]))

Answer:

B

Question 5

Question Type: **FillInTheBlank**

Problem Scenario 47 : You have been given below code snippet, with intermediate output.

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

// lets first print out the contents of the RDD with partition labels

def myfunc(index: Int, iter: Iterator[(Int)]): Iterator[String] = {

iter.toList.map(x => "[partID:" + index + ", val: " + x + "]").iterator

}

//In each run , output could be different, while solving problem assume belowm output only.

z.mapPartitionsWithIndex(myfunc).collect

res28: Array[String] = Array([partID:0, val: 1], [partID:0, val: 2], [partID:0, val: 3], [partID:1, val: 4], [partID:1, val: S], [partID:1, val: 6])

Now apply aggregate method on RDD z , with two reduce function , first will select max value in each partition and second will add all the maximumvalues from all partitions.

Initialize the aggregate with value 5. hence expected output will be 16.

Answer:

Question 6

Question Type: MultipleChoice

Problem Scenario 46 : You have been given belwo list in scala (name,sex,cost) for each work done.

List(("Deeapak" , "male", 4000), ("Deepak" , "male", 2000), ("Deepika" , "female", 2000),("Deepak" , "female", 2000), ("Deepak" , "male", 1000) , ("Neeta" , "female", 2000))

Now write a Spark program to load this list as an RDD and do the sum of cost for combination of name and sex (as key)

Options:

A- Solution :

Step 1 : Create an RDD out of this list
val rdd = sc.parallelize(List(('Deeapak' , 'male', 4000}, ('Deepak' , 'male', 2000), ('Deepika' , 'female', 2000),('Deepak' , 'female', 2000),('Deepak' , 'male', 1000} , ('Neeta' , 'female', 2000}}}
Step 2 : Convert this RDD in pair RDD
val byKey = rdd.map({case (name,sex,cost) => (name,sex)->cost})
Step 3 : Now group by Key
val byKeyGrouped = byKey.groupByKey
Step 4 : Nowsum the cost for each group
val result = byKeyGrouped.map{case ((id1,id2),values) => (id1,id2,values.sum)}
Step 5 : Save the results result.repartition(1).saveAsTextFile('spark12/result.txt')

B- Solution :
Step 1 : Create an RDD out of this list
val rdd = sc.parallelize(List(('Deeapak' , 'male', 2000}, ('Deepak' , 'male', 2000), ('Deepika' , 'female', 6000),('Deepak' , 'female', 6000),('Deepak' , 'male', 1000} , ('Neeta' , 'female', 2000}}}
Step 2 : Convert this RDD in pair RDD
val byKey = rdd.map({case (name,sex,cost) => (name,sex)->cost})
Step 3 : Now group by Key
val byKeyGrouped = byKey.groupByKey
Step 4 : Nowsum the cost for each group
val result = byKeyGrouped.map{case ((id1,id2),values) => (id1,id2,values.sum)}
Step 5 : Save the results result.repartition(1).saveAsTextFile('spark12/result.txt')

Answer:

A

Question 7

Question Type: **MultipleChoice**

Problem Scenario 45 : You have been given 2 files , with the content as given Below

(spark12/technology.txt)

(spark12/salary.txt)

(spark12/technology.txt)

first,last,technology

Amit,Jain,java

Lokesh,kumar,unix

Mithun,kale,spark

Rajni,vekat,hadoop

Rahul,Yadav,scala

(spark12/salary.txt)

first,last,salary

Amit,Jain,100000

Lokesh,kumar,95000

Mithun,kale,150000

Rajni,vekat,154000

Rahul,Yadav,120000

Write a Spark program, which will join the data based on first and last name and save the joined results in following format, first Last.technology.salary

Options:

A- Solution :
Step 1 : Create 2 files first using Hue in hdfs.
Step 2 : Load all file as an RDD
val technology = sc.textFile(Msparkl2/technology.txt').map(e =>e.splitf','))
val salary = sc.textFile('spark12/salary.txt').map(e => e.split('.'))
Step 3 : Save the results in a text file as below.
joined.repartition(1).saveAsTextFile('spark12/multiColumn Joined.txt')

B- Solution :
Step 1 : Create 2 files first using Hue in hdfs.

Step 2 : Load all file as an RDD
val technology = sc.textFile(Msparkl2/technology.txt').map(e =>e.splitf','))
val salary = sc.textFile('spark12/salary.txt').map(e => e.split('.'))
Step 3 : Now create Key.value pair of data and join them.
val joined = technology.map(e=>((e(0),e(1)),e(2))).join(salary.map(e=>((e(0),e(1)),e(2))))
Step 4 : Save the results in a text file as below.
joined.repartition(1).saveAsTextFile('spark12/multiColumn Joined.txt')

Answer:

B

Question 8

Question Type: **MultipleChoice**

Problem Scenario 44 : You have been given 4 files , with the content as given below:

spark11/file1.txt

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework

spark11/file2.txt

The core of Apache Hadoop consists of a storage part known as Hadoop Distributed File System (HDFS) and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed.

spark11/file3.txt

his approach takes advantage of data locality nodes manipulating the data they have access to to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking

spark11/file4.txt

Apache Storm is focused on stream processing or what some call complex event processing. Storm implements a fault tolerant method for performing a computation or pipelining multiple computations on an event as it flows into a system. One might use Storm to transform unstructured data as it flows into a system into a desired format

(spark11Afile1.txt)

(spark11/file2.txt)

(spark11/file3.txt)

(sparkl 1/file4.txt)

Write a Spark program, which will give you the highest occurring words in each file. With their file name and highest occurring words.

Options:

A- Solution :

Step 1 : Create all 4 file first using Hue in hdfs.

Step 2 : Load all file as an RDD

val file1 = sc.textFile('sparkl1/filel.txt')

val file2 = sc.textFile('spark11/file2.txt')

val file3 = sc.textFile('spark11/file3.txt')

val file4 = sc.textFile('spark11/file4.txt')

Step 3 : Now do the word count for each file and sort in reverse order of count.

val contentl = filel.flatMap(line => line.split(' ')).map(word => (word,1)).reduceByKey(_ + _).map(item => item.swap).sortByKey(false).map(e=>e.swap)

val content.2 = file2.flatMap(line => line.splitf ')).map(word => (word,1)).reduceByKey(_ + _).map(item => item.swap).sortByKey(false).map(e=>e.swap)

val content3 = file3.flatMap(line > line.split)' ')).map(word => (word,1)).reduceByKey(_ + _).map(item => item.swap).sortByKey(false).map(e=>e.swap)

val content4 = file4.flatMap(line => line.split(' ')).map(word => (word,1)).reduceByKey(_ + _).map(item => item.swap).sortByKey(false).map(e=>e.swap)

Step 4 : Split the data and create RDD of all Employee objects.

val filelword = sc.makeRDD(Array(file1.name+'->'+content1(0)._1+'-'+content1(0)._2)) val file2word = sc.makeRDD(Array(file2.name+'->'+content2(0)._1+'-'+content2(0)._2)) val file3word = sc.makeRDD(Array(file3.name+'->'+content3(0)._1+'-'+content3(0)._2)) val file4word = sc.makeRDD(Array(file4.name+M->'+content4(0)._1+'-'+content4(0)._2))

Step 5: Union all the RDDS

val unionRDDs = filelword.union(file2word).union(file3word).union(file4word)

Step 6 : Save the results in a text file as below. unionRDDs.repartition(1).saveAsTextFile('spark11/union.txt')

B- Solution :

Step 1 : Create all 4 file first using Hue in hdfs.

Step 2 : Load all file as an RDD

val file1 = sc.textFile('sparkl1/filel.txt')

val file2 = sc.textFile('spark11/file2.txt')

val file3 = sc.textFile('spark11/file3.txt')

val file4 = sc.textFile('spark11/file4.txt')

Step 3 : Now do the word count for each file and sort in reverse order of count.

val contentl = filel.flatMap(line => line.split(' ')).map(word => (word,1)).reduceByKey(_ + _).map(item => item.swap).sortByKey(false).map(e=>e.swap)

val content.2 = file2.flatMap(line => line.splitf ')).map(word => (word,1)).reduceByKey(_ + _).map(item => item.swap).sortByKey(false).map(e=>e.swap)

val content3 = file3.flatMap(line > line.split)' ')).map(word => (word,1)).reduceByKey(_ + _).map(item => item.swap).sortByKey(false).map(e=>e.swap)

val content4 = file4.flatMap(line => line.split(' ')).map(word => (word,1)).reduceByKey(_ + _).map(item => item.swap).sortByKey(false).map(e=>e.swap)

Step 4: Union all the RDDS

val unionRDDs = filelword.union(file2word).union(file3word).union(file4word)

Step 5 : Save the results in a text file as below. unionRDDs.repartition(1).saveAsTextFile('spark11/union.txt')

Answer:

A

Question 9

Question Type: MultipleChoice

Problem Scenario 43 : You have been given following code snippet.

val grouped = sc.parallelize(Seq(((1,"twoM), List((3,4), (5,6)))))

val flattened = grouped.flatMap {A =>

groupValues.map { value => B }

}

You need to generate following output.

Hence replace A and B

Array((1,two,3,4),(1,two,5,6))

Options:

A- Solution :
A case (key, groupValues)
B (key._1, key._2, value._1, value._2)

B- Solution :

A case (key, groupValues)

B (key._1, key._3, value._2, value._2)

Answer:

A

Question 10

Question Type: **MultipleChoice**

Problem Scenario 42 : You have been given a file (sparklO/sales.txt), with the content as given in below.

spark10/sales.txt

Department,Designation,costToCompany,State

Sales,Trainee,12000,UP

Sales,Lead,32000,AP

Sales,Lead,32000,LA

Sales,Lead,32000,TN

Sales,Lead,32000,AP

Sales,Lead,32000,TN

Sales,Lead,32000,LA

Sales,Lead,32000,LA

Marketing,Associate,18000,TN

Marketing,Associate,18000,TN

HR,Manager,58000,TN

And want to produce the output as a csv with group by Department,Designation,State with additional columns with sum(costToCompany)and TotalEmployeeCountt

Should get result like

Dept,Desg,state,empCount,totalCost

Sales,Lead,AP,2,64000

Sales.Lead.LA.3.96000

Sales,Lead,TN,2,64000

Options:

A- Solution :

step 1 : Create a file first using Hue in hdfs.
Step 2 : Load tile as an RDD
val rawlines = sc.textFile('spark10/sales.txt')
Step 3 : Create a case class, which can represent its column fileds. case class Employee(dep: String, des: String, cost: Double, state: String)
Step 4 : Split the data and create RDD of all Employee objects.
val employees = rawlines.map(_.split(',')).map(row=>Employee(row(0), row{1), row{2).toDouble, row{3)))
Step 5 : Create a row as we needed. All group by fields as a key and value as a count for each employee as well as its cost, val keyVals = employees.map(em => ((em.dep, em.des, em.state), (1 , em.cost)))
Step 6 : Group by all the records using reduceByKey method as we want summation as well. For number of employees and their total cost, val results = keyVals.reduceByKey{ (a,b) => (a._1 + b._1, a._2 + b._2)} // (a.count + b.count, a.cost + b.cost)}
Step 7 : Save the results in a text file as below. results.repartition(1).saveAsTextFile('spark10/group.txt')

B- Solution :
step 1 : Create a file first using Hue in hdfs.
Step 2 : Load tile as an RDD
val rawlines = sc.textFile('spark10/sales.txt')
Step 3 : Create a case class, which can represent its column fileds. case class Employee(dep: String, des: String, cost: Double, state: String)
Step 4 : Split the data and create RDD of all Employee objects.
val employees = rawlines.map(_.split(',')).map(row=>Employee(row(0), row{1), row{2).toDouble, row{3)))
Step 5 : Group by all the records using reduceByKey method as we want summation as well. For number of employees and their total cost, val results = keyVals.reduceByKey{ (a,b) => (a._1 + b._1, a._2 + b._2)} // (a.count + b.count, a.cost + b.cost)}
Step 7 : Save the results in a text file as below. results.repartition(1).saveAsTextFile('spark10/group.txt')

Answer:

A