CSCE 5013 Cloud Computing
Final Exam, Spring 2018 (Total 10 pages)

Name:________________________, ID:_____________________

Totally, 100 points.

Problem Set #1 (30 points) (2 points each)

1.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4))
val result = sourcerdd.fold(0)((x,y)=>x+y)
```
What is the content of `result`?

Solution:

2.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4))
val result = sourcerdd.fold(1)((x,y)=>x+y)
```
What is the content of `result`?

Solution:

3.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val result = sourcerdd.fold(1)((x,y)=>x+y)
```
What is the content of `result`?

Solution:

4.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val result = sourcerdd.fold(0)((x,y)=>x+1)
```
What is the content of `result`?

Solution:

5.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val result = sourcerdd.fold(1)((x,y)=>x+1)
```
What is the content of `result`?

Solution:

6.
```
val rdd1 = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val rdd2 = rdd1.repartition(10)
val result = rdd2.fold(0)((x,y)=>x+1)
```
What is the content of `result`?

Solution:

7.
```
val rdd1 = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val rdd2 = rdd1.coalesce(10)
val result = rdd2.fold(0)((x,y)=>x+1)
```
What is the content of `result`?

Solution:

8.
```
val rdd1 = sc.parallelize(List(-1, -2, 1, 2, 3, 4),6)
val result = rdd1.aggregate(1)((x, y) => x + 1, (x, y) => x + 1)
```
What is the content of `result`?

Solution:

9.
```
val rdd1 = sc.parallelize(List((1,2), (3,4), (3,6)))
val result = rdd1.mapValues(x=>x/2).map(x=>x._1*x._2).reduce((x,y)=>x+y)
```
What is the content of `result`?

Solution:

10.
```
val sourcerdd = sc.parallelize(List((1,2),(1,3),(1,2),(2,6),(3,5),
 (3,6),(2,6),(4,10)),2)
val result = sourcerdd.reduceByKey((x,y)=>x+y)
```
What is the content of `result`?

Solution:

11.
```
val sourcerdd = sc.parallelize(List((1,2),(1,3),(1,2),(2,6),(3,5),
 (3,6),(2,6),(4,10)),2)
val result = sourcerdd.reduceByKey((x,y)=>x+1)
```
What is the content of `result`?

Solution:

12.
```
val sourcerdd = sc.parallelize(List((1,2),(1,3),(1,2),(2,6),(3,5),
(3,6),(2,6),(4,10)),2)
val result = sourcerdd.foldByKey(0)((x,y)=>x+y)
```
What is the content of `result`?

Solution:

13.
```
val sourcerdd = sc.parallelize(List((1,2),(1,3),(1,2), (2,6),(3,5),
(3,6),(2,6),(4,10)),2)
val result = sourcerdd.foldByKey(1)((x,y)=>x+1)
```
What is the content of `result`?

Solution:

14.
```
val sourcerdd = sc.parallelize(List (1,2),(1,3),(1,2),(2,6),(3,5),
(3,6),(2,6),(4,10),(3,10)),3)
val result = sourcerdd.foldByKey(1)((x,y)=>x+1)
```
What is the content of `result`?

Solution:

15.
```
val x = sc.parallelize(List(("a", 1), ("b", 1), ("a", 2), ("b", 3)))
val result = x.combineByKey((v) => (v, 2),(acc: (Int, Int), v) => (acc._1
+ v, acc._2 + 1),(acc1: (Int, Int), acc2: (Int, Int)) => (acc1._1 +
acc2._1, acc1._2 + acc2._2)).mapValues(value => value._1 /
value._2.toFloat)
```
What is the content of `result`?

Solution:

Problem Set #2 (20 points) Write the pseudo code to solve problem in MapReduce. You need to specify what are the keys and the corresponding values for the inputs, the intermediate results, and the outputs.

The input is a very large file of terabytes. The file contains lines, each of which consists of words delimited by space**.** Following is the example of the first 10 lines.

```
This is
an arbitrary example file
of 10 lines
Each line does
not have to be
of
the same
length or contain
the same
number of words
```

You need to implement a MapReduce application to generate the index for words in the file, i.e., the offsets of a word from the beginning of the file. If a word has multiple presences in the file, all offsets need to be listed. The following lists the output of two words using the 10 lines as an example.

```
This 0
Of 32 71
```

Assuming that you can call a function `findtheoffset(line)`, which will return an array of local offsets of all words in a line. For example, given a line of "`the same`", it will return an array of (0, 4).

(1) (10 points) In the first implementation, the multiple offsets of the same word can be in a random order.

(2) (10 points) In the second implementation, the offsets of the same word need to be in an ascending order. Your implementation needs to be scalable, i.e., do not perform sorting in the Reducer task.

Problem Set #3 (20 points) Secondary sorting.

Following is a log file recording the reading of many sensors in time sequence.

```
(t1,m1,r11) #t1: time, m1: sensor ID, r11: reading
(t1,m2,r12)
  ……
(t2,m1,r21)
(t2,m2,r22)
```

After processing, the reading of a particular sensor needs to be in an order as follows.

```
(m1,t1,r11)
(m1,t2,r21)
(m1,t3,r31)
  ……
```

(1) (10 points) Design a MapReduce program in pseudo code to implement the secondary sorting. Do not perform sorting in the Reducer task.

(2) (10 points) Design a Spark program in pseudo code to implement the secondary sorting. Use pair RDD in your implementation. Specify the transformations and actions in your implementation.

Problem #4 (10 points) Spark programming. Specify the transformations and actions in your implementation. No pseudo code.

(1) (5 points) Calculate the per-key average in pair RDD. The example is as follows.

Key	**Value**
panda	0
pink	3
pirate	3
panda	1
pink	4

→

Key	**Average Value**
panda	0.5
pink	3.5
pirate	3

(2) (5 points) Count bigrams in a file (i.e., input.txt), which contains multiple lines. Each single line consists of multiple words that are separated by space. A bigram is two consecutive words in the same line.

You can use sliding(`n`) transformation that will slide down the sub_items in an item of RDD with a window of size of `n`. The `n` sub_items in the window will be returned as a new sub_item of an item in the new RDD. For example,

```
rdd=((a,b,c),(d,e),(f,g,h,i),(j))
rdd.sliding(2)=(((a,b),(b,c)),((d,e)),((f,g),(g,h),(h,i)),())
```

Problem #5 (10 points) Given the content of the input file, which specifies the links in a graph (i.e., (a, b) represents a link from node a to node b), and the Spark code, specify the page rank mass of those nodes after two iterations.

The input file:
1 2
1 4
2 1
2 3
3 1
3 4
3 5

```
val sparkConf = new SparkConf().setAppName("PageRank")
val iters = if (args.length > 0) args(0).toInt else 10
val sc = new SparkContext(sparkConf)

val lines = sc.textFile("input.txt")
val links = lines.map{ s =>
               val parts = s.split("\\s+")
               (parts(0), parts(1))
        }.distinct().groupByKey()
val nodes=lines.flatMap(line => line.split(" ")).distinct()
val danglingNodes=nodes.subtract(links.keys).collect()
val linkList = links ++ sc.parallelize(for (i <- danglingNodes) yield (i,
List.empty))
val nodeSize = linkList.count()
        var ranks = linkList.mapValues(v => 1.0 / nodeSize)
        for (i <- 1 to iters) {
            val dangling = sc.accumulator(0.0)
            val contribs = linkList.join(ranks).values.flatMap {
                case (pageLinks, rank) => {
                    val size = pageLinks.size
                    if (size == 0) {
                        dangling += rank
                        List()
                    } else {
                        pageLinks.map(pageLink => (pageLink, rank / size))
                    }
                }
            }
            contribs.count()
            val danglingValue = dangling.value
            ranks = contribs.reduceByKey(_ + _, 3 ).mapValues[Double](p =>
                0.1 * (1.0 / nodeSize) + 0.9 * (danglingValue / nodeSize +
p))
        }
ranks.sortBy(_._1, true).saveAsTextFile("output")
```

Node ID	Page Rank Mass after 2 iterations (keep 3 digits after the decimal point, e.g., 1.334)
1	
2	
3	
4	
5	

Problem #6 (10 points) Given the Spark code below, specify the partitioners and the number of partitions associated with RDDs.

```
import org.apache.spark.HashPartitioner
//assuming that there are thousands of pair elements in rdd1
val rdd1 = sc.parallelize(List(("a", 1), ("b", 1), ("a", 2), ...., ("ef", 2)), 100)
val rdd2 = rdd1.repartition(50)
val rdd3 = rdd2.partitionBy(new HashPartitioner(100))
val rdd4 = rdd3.map(x=>x)
val rdd5 = rdd4.reduceByKey((x,y)=>x+y)
```

RDD	**Partitioner**	**Number of Partitions**
rdd1		
rdd2		
rdd3		
rdd4		
rdd5		

CSCE 5013 Cloud Computing
Final Exam, Spring 2018 (Total 10 pages)

Name:_________________________, ID:______________________

<u>Totally, 100 points.</u>

Problem Set #1 (30 points) (2 points each)

1.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4))
val result = sourcerdd.fold(0)((x,y)=>x+y)
```
What is the content of `result`?

Solution: 7

2.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4))
val result = sourcerdd.fold(1)((x,y)=>x+y)
```
What is the content of `result`?

Solution: 9

3.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val result = sourcerdd.fold(1)((x,y)=>x+y)
```
What is the content of `result`?

Solution: 12

4.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val result = sourcerdd.fold(0)((x,y)=>x+1)
```
What is the content of `result`?

Solution: 4

5.
```
val sourcerdd = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val result = sourcerdd.fold(1)((x,y)=>x+1)
```
What is the content of `result`?

Solution: 5

6.
```
val rdd1 = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val rdd2 = rdd1.repartition(10)
val result = rdd2.fold(0)((x,y)=>x+1)
```
What is the content of `result`?

Solution: 10

7.
```
val rdd1 = sc.parallelize(List(-1, -2, 1, 2, 3, 4),4)
val rdd2 = rdd1.coalesce(10)
val result = rdd2.fold(0)((x,y)=>x+1)
```
What is the content of `result`?

Solution: 4

8.
```
val rdd1 = sc.parallelize(List(-1, -2, 1, 2, 3, 4),6)
val result = rdd1.aggregate(1)((x, y) => x + 1, (x, y) => x + 1)
```
What is the content of `result`?

Solution: 7

9.
```
val rdd1 = sc.parallelize(List((1,2), (3,4), (3,6)))
val result = rdd1.mapValues(x=>x/2).map(x=>x._1*x._2).reduce((x,y)=>x+y)
```
What is the content of `result`?

Solution: 16

10.
```
val sourcerdd = sc.parallelize(List((1,2),(1,3),(1,2),(2,6),(3,5),
 (3,6),(2,6),(4,10)),2)
val result = sourcerdd.reduceByKey((x,y)=>x+y)
```
What is the content of `result`?

Solution: (1,7), (2,12), (3,11), (4,10)

11.
```
val sourcerdd = sc.parallelize(List((1,2),(1,3),(1,2),(2,6),(3,5),
 (3,6),(2,6),(4,10)),2)
val result = sourcerdd.reduceByKey((x,y)=>x+1)
```
What is the content of `result`?

Solution: (1,4), (2,7), (3,6), (4,10)

12.
```
val sourcerdd = sc.parallelize(List((1,2),(1,3),(1,2),(2,6),(3,5),
(3,6),(2,6),(4,10)),2)
val result = sourcerdd.foldByKey(0)((x,y)=>x+y)
```
What is the content of `result`?

Solution: (1,7), (2,12), (3,11), (4,10)

13.
```
val sourcerdd = sc.parallelize(List((1,2),(1,3),(1,2), (2,6),(3,5),
(3,6),(2,6),(4,10)),2)
val result = sourcerdd.foldByKey(1)((x,y)=>x+1)
```
What is the content of `result`?

Solution: (1,4), (2,3), (3,3), (4,2)

14.
```
val sourcerdd = sc.parallelize(List (1,2),(1,3),(1,2),(2,6),(3,5),
(3,6),(2,6),(4,10),(3,10)),3)
val result = sourcerdd.foldByKey(1)((x,y)=>x+1)
```
What is the content of `result`?

Solution: (1,4), (2,3), (3,4), (4,2)

15.
```
val x = sc.parallelize(List(("a", 1), ("b", 1), ("a", 2), ("b", 3)))
val result = x.combineByKey((v) => (v, 2),(acc: (Int, Int), v) => (acc._1
+ v, acc._2 + 1),(acc1: (Int, Int), acc2: (Int, Int)) => (acc1._1 +
acc2._1, acc1._2 + acc2._2)).mapValues(value => value._1 /
value._2.toFloat)
```
What is the content of `result`?

Solution: ((a, 1.0), (b, 1.333))

Problem Set #2 (20 points) Write the pseudo code to solve problem in MapReduce. You need to specify what are the keys and the corresponding values for the inputs, the intermediate results, and the outputs.

The input is a very large file of terabytes. The file contains lines, each of which consists of words delimited by space**.** Following is the example of the first 10 lines.

```
This is
an arbitrary example file
of 10 lines
Each line does
not have to be
of
the same
length or contain
the same
number of words
```

You need to implement a MapReduce application to generate the index for words in the file, i.e., the offsets of a word from the beginning of the file. If a word has multiple presences in the file, all offsets need to be listed. The following lists the output of two words using the 10 lines as an example.

```
This 0
Of 32 71
```

Assuming that you can call a function `findtheoffset(line)`, which will return an array of local offsets of all words in a line. For example, given a line of "`the same`", it will return an array of (0, 4).

(1) (10 points) In the first implementation, the multiple offsets of the same word can be in a random order.

```
Mapper{
      map(int offset, text line) {
            Wordarray = texttotoken(line);
            Offsetarray = findtheoffset(line);
            for (i=0; i<Wordarray.length; i++)
                  emit(Wordarray[i], Offsetarray[i]+offset);
      }
}




Reducer{
      reduce(word, [offset_1, offset_2, …]){
            for (offset ϵ [offset_1, offset_2, …])
                  offsets.append(offset);
            emit(word, offsets);
      }
}
```

(2) (10 points) In the second implementation, the offsets of the same word need to be in an ascending order. Your implementation needs to be scalable, i.e., do not perform sorting in the Reducer task.

```
1. Implement a word_offset class for the data structure (word, offset). Two
methods need to be implemented for this class.
      (1). Generate the hash value of (word, offset) based on the word only.
      (2). Implement the comparison between (word, offset)_1 and
(word, offset)_2 firstly on word, then on offset.

2.
Mapper{
      map(int offset, text line) {
            Wordarray = texttotoken(line);
            Offsetarray = findtheoffset(line);
            for (i=0; i<Wordarray.length; i++)
                  emit((Wordarray[i], Offsetarray[i]+offset),1);
      }
}

3.
Reducer{
      reduce((word, offset),1){
            if (word == previous_word)
                  offsets.append(offset);
            else {
                  emit(word, offsets);
                  previous_word=word;
            }
      }

      close{
            emit(word,offsets);
      }
}
```

Problem Set #3 (20 points) Secondary sorting.

Following is a log file recording the reading of many sensors in time sequence.

```
        (t1,m1,r11) #t1: time, m1: sensor ID, r11: reading
        (t1,m2,r12)
          ……
        (t2,m1,r21)
        (t2,m2,r22)
```

After processing, the reading of a particular sensor needs to be in an order as follows.

```
        (m1,t1,r11)
        (m1,t2,r21)
        (m1,t3,r31)
          ……
```

(1) (10 points) Design a MapReduce program in pseudo code to implement the secondary sorting. Do not perform sorting in the Reducer task.

```
1. Implement a sensor_time class for the data structure (m, t). Two methods
need to be implemented for this class.
      (1). Generate the hash value of (m, t) based on m only.
      (2). Implement the comparison between (m, t)_1 and (m, t)_2 firstly on
m, then on t.

2.
Mapper{
      map(int offset, text line) {
            array = texttotoken(line);
            m = array[1];
            t = array[0];
            r = array[2];
            emit((m, t), r);
      }
}

3.
Reducer{
      reduce((m, t), r){
            emit((m, t), r);
      }
}
```

(2) (10 points) Design a Spark program in pseudo code to implement the secondary sorting. Use pair RDD in your implementation. Specify the transformations and actions in your implementation.

(1) Create the input RDD by reading log records from the file. If we assume that the original log is saved in text format, then use textFile() method.

```
val rdd1=sc.textFile(input.txt)
```

(2) Convert rdd1 to pair RDD.

```
val rdd2=rdd1.map(x => x.split(",")).map(x=>((x._2,x._1),x._3))
```

(3) Sort rdd2 using sortByKey() transformation. Define the own comparison function in which first compare the m and then compare the t in the key.

```
val rdd3=rdd2.sortByKey()
```

Problem #4 (10 points) Spark programming. Specify the transformations and actions in your implementation. No pseudo code.

(1) (5 points) Calculate the per-key average in pair RDD. The example is as follows.

Key	Value
panda	0
pink	3
pirate	3
panda	1
pink	4

→

Key	Average Value
panda	0.5
pink	3.5
pirate	3

```
val result =
rdd.mapValues(x=>(x,1)).reduceByKey((x,y)=>(x._1+y._1,x._2+y._2)).
mapValues(x=>x._1/x._2.toFloat)
```

(2) (5 points) Count bigrams in a file (i.e., input.txt), which contains multiple lines. Each single line consists of multiple words that are separated by space. A bigram is two consecutive words in the same line.

You can use sliding(n) transformation that will slide down the sub_items in an item of RDD with a window of size of n. The n sub_items in the window will be returned as a new sub_item of an item in the new RDD. For example,

```
rdd=((a,b,c),(d,e),(f,g,h,i),(j))
rdd.sliding(2)=(((a,b),(b,c)),((d,e)),((f,g),(g,h),(h,i)),())
```

```
val input=sc.textFile("input.txt")
val bigrams=input.Map(x=>x.split(" ")).sliding(2).flatMap(x=>x)
val result=bigrams.map(x=>(x,1)).reduceByKey((x,y)=>x+y)
```

Problem #5 (10 points) Given the content of the input file, which specifies the links in a graph (i.e., (a, b) represents a link from node a to node b), and the Spark code, specify the page rank mass of those nodes after two iterations.

The input file: 1 2
1 4
2 1
2 3
3 1
3 4
3 5

```
val sparkConf = new SparkConf().setAppName("PageRank")
val iters = if (args.length > 0) args(0).toInt else 10
val sc = new SparkContext(sparkConf)

val lines = sc.textFile("input.txt")
val links = lines.map{ s =>
               val parts = s.split("\\s+")
               (parts(0), parts(1))
        }.distinct().groupByKey()
val nodes=lines.flatMap(line => line.split(" ")).distinct()
val danglingNodes=nodes.subtract(links.keys).collect()
val linkList = links ++ sc.parallelize(for (i <- danglingNodes) yield (i,
List.empty))
val nodeSize = linkList.count()
        var ranks = linkList.mapValues(v => 1.0 / nodeSize)
        for (i <- 1 to iters) {
            val dangling = sc.accumulator(0.0)
            val contribs = linkList.join(ranks).values.flatMap {
                case (pageLinks, rank) => {
                    val size = pageLinks.size
                    if (size == 0) {
                        dangling += rank
                        List()
                    } else {
                        pageLinks.map(pageLink => (pageLink, rank / size))
                    }
                }
            }
            contribs.count()
            val danglingValue = dangling.value
            ranks = contribs.reduceByKey(_ + _, 3 ).mapValues[Double](p =>
                0.1 * (1.0 / nodeSize) + 0.9 * (danglingValue / nodeSize +
p))
        }
ranks.sortBy(_._1, true).saveAsTextFile("output")
```

Node ID	Page Rank Mass after 2 iterations (keep 3 digits after the decimal point, e.g., 1.334)
1	0.227
2	0.200
3	0.173
4	0.254
5	0.146

Problem #6 (10 points) Given the Spark code below, specify the partitioners and the number of partitions associated with RDDs.

```
import org.apache.spark.HashPartitioner
//assuming that there are thousands of pair elements in rdd1
val rdd1 = sc.parallelize(List(("a", 1), ("b", 1), ("a", 2), ...., ("ef", 2)), 100)
val rdd2 = rdd1.repartition(50)
val rdd3 = rdd2.partitionBy(new HashPartitioner(100))
val rdd4 = rdd3.map(x=>x)
val rdd5 = rdd4.reduceByKey((x,y)=>x+y)
```

RDD	**Partitioner**	**Number of Partitions**
rdd1	None	100
rdd2	None	50
rdd3	HashPartitioner(100)	100
rdd4	None	100
rdd5	HashPartitioner(100)	100