سوالات تمرین شماره سه BIG DATA

تاریخ تحویل : ۹۸/۹/۳۰ نمره : ۲

---

۱. پایگاه داده FLIGHT که شامل اطلاعات پروازي مربوط به یکسال فرودگاه هاي امریکا است در پیوست آورده شده است. میخواهیم با استفاده از کتابخانه GRAPHX در SPARK موارد زیر را از روي این پایگاه داده استخراج نماییم.

توضیحات در مورد پارس کردن دیتا:

از آنجایی که فرمت فایل به صورت csv بود ابتدا در اکسل تنها به فایل ها یک هدر اضافه شد سپس به صورت dataframe در آورده شد و در مسائل مورد استفاده قرار گرفت. فایل اطلاعات در ضمیمه قرار گرفت.

a: تعداد کل فرودگاهها و همینطور مسیرهاي ارتباطي نقطه به نقطه را مشخص کنید.

تعداد کل فرود گاه ها ۵۱ است.

```
+---+----+
| id|node|
+---+----+
|  0| DLG|
|  1| GEG|
|  2| SNA|
|  3| BUR|
|  4| PSG|
|  5| OAK|
|  6| SCC|
|  7| DCA|
|  8| WRG|
|  9| KTN|
| 10| CDV|
| 11| ADK|
```

12	LIH
13	HNL
14	GST
15	SJC
16	LGB
17	RNO
18	BOS
19	EWR
20	LAS
21	FAI
22	DEN
23	OME
24	PSP
25	BOI
26	SEA
27	PDX
28	MIA
29	SMF
30	BRW
31	PHX
32	BET
33	DFW
34	SFO
35	AKN
36	ORD
37	TUS
38	JNU
39	KOA
40	ADQ
41	ONT
42	LAX
43	MSP
44	SIT
45	MCO
46	OTZ
47	SAN
48	YAK
49	ANC
50	OGG
+---+----+

و تعداد مسیر های ارتباطی بین این فرودگاه ها ۱۷۰ است و فاصله بین ها نیز مشخص شده:

orgin_	dest_	distance
10	49	160
26	13	2677
44	38	95
22	49	2406
49	26	1449
27	45	2534
27	31	1009
49	6	626
0	49	329
26	22	1024
42	27	834
26	34	679
29	27	479
9	38	234
31	27	1009
26	50	2640
38	14	41
26	19	2401
26	45	2553
26	3	937
21	6	373
32	49	399
34	27	550
40	49	252
16	26	965
26	2	978
48	10	213
26	12	2701
26	47	1050
26	41	956
38	26	909
18	27	2537
49	13	2777
20	26	866
49	36	2846
47	26	1050

24	34	421
45	27	2534
22	27	992
38	49	571
46	23	183
26	33	1660
19	26	2401
26	20	866
27	42	834
13	26	2677
27	47	933
34	24	421
27	22	992
24	26	987
49	46	549
5	27	543
39	26	2688
26	44	862
30	21	503
27	20	762
45	26	2553
38	44	95
27	3	817
49	35	289
34	42	337
26	9	680
26	39	2688
24	27	873
26	21	1533
18	26	2496
29	26	605
36	49	2846
49	11	1192
26	25	399
44	9	183
17	26	564
6	49	626
43	26	1399
26	31	1107
21	49	261
36	26	1721

26	15	697
49	40	252
4	8	31
27	34	550
49	32	399
38	9	234
1	26	224
27	24	873
9	8	82
26	43	1399
26	42	954
2	26	978
10	48	213
49	10	160
49	22	2406
26	18	2496
34	49	2018
27	2	859
27	15	569
42	7	2311
6	30	204
49	0	329
21	30	503
42	26	954
3	26	937
41	27	838
26	29	605
49	21	261
8	4	31
42	34	337
4	38	123
3	27	817
26	24	987
23	49	539
31	26	1107
2	5	371
48	38	199
49	38	571
8	9	82
23	46	183
5	2	371

5	26	671
37	26	1216
47	27	933
49	50	2797
42	49	2345
26	36	1721
25	26	399
26	38	909
12	26	2701
35	0	71
7	26	2329
38	48	199
26	28	2724
9	44	183
26	49	1449
7	42	2311
26	17	564
34	26	679
27	5	543
11	49	1192
35	49	289
49	23	539
13	49	2777
27	29	479
28	26	2724
27	41	838
20	27	762
46	49	549
49	34	2018
9	26	680
26	37	1216
15	27	569
22	26	1024
15	26	697
50	49	2797
33	26	1660
30	49	725
21	26	1533
49	27	1542
14	38	41
27	18	2537

```
|    26|    5|     671|
|    26|    7|    2329|
|    38|    4|     123|
|    44|   26|     862|
|    26|   16|     965|
|    49|   42|    2345|
|    27|   49|    1542|
|    50|   26|    2640|
|     2|   27|     859|
|    26|    1|     224|
|    41|   26|     956|
+------+-----+--------+
```

b:طولاني ترين مسير و مسيرهاي بالاي ۱۰۰۰ مايل را مشخص كنيد.

۶۰ مسیر بیش از ۱۰۰۰ مایل و ۱۰ مسیر برتر شامل زیر می‌شود.

```
+------+-----+--------+
|orgin_|dest_|distance|
+------+-----+--------+
|    49|   36|    2846|
|    36|   49|    2846|
|    49|   50|    2797|
|    50|   49|    2797|
|    13|   49|    2777|
|    49|   13|    2777|
|    26|   28|    2724|
|    28|   26|    2724|
|    12|   26|    2701|
|    26|   12|    2701|
+------+-----+--------+
only showing top 10 rows
```

خروجی با فرمت triplets

```
Distance 2846 from ANC to ORD.

Distance 2846 from ORD to ANC.

Distance 2797 from ANC to OGG.
```

```
Distance 2797 from OGG to ANC.

Distance 2777 from ANC to HNL.

Distance 2777 from HNL to ANC.

Distance 2724 from SEA to MIA.

Distance 2724 from MIA to SEA.

Distance 2701 from SEA to LIH.
```

c : فرودگاه هاي با بيشترين تعداد مسيرهاي ورودي و بيشترين تعداد مسيرهاي خروجي را مشخص نماييد.

فرودگاه ها با بيشترين مسير خروجی شامل :

```
+---------+----+
|node|output-ord|
+---------+----+
|SEA|        35|
|ANC|        19|
|PDX|        16|
|JNU|         7|
|SFO|         5|
|LAX|         5|
|KTN|         4|
|FAI|         4|
+---------+----+
only showing top 8 rows
```

فرودگاه ها با بيشترين مسير داخلی شامل :

```
+----+---------+
|node|input-ord|
+----+---------+
| SEA|       35|
| ANC|       20|
| PDX|       16|
| JNU|        7|
| SFO|        5|
| LAX|        5|
| KTN|        4|
```

```
| OAK|        3|
+----+---------+
only showing top 8 rows
```

همچنین میتوان با استفاده از توابع خود GraphX درجه نقاط را محاسبه نمود که :

```
maxInDegree :SEA,35 maxOutDegree :SEA,35 maxDegrees :SEA,70
```

d: چه فرودگاهي به عنوان مهمترين هاب (نشستن و برخاستن موقت) مطرح است.

برای این کار می بایست یک معیار ریاضی تعریف کنیم که هاب بودن(هابیت) را تشخیص دهیم برای اینکار تعداد رابطه زیر استفاده شد

$$\text{تعداد نودها خروجی} \times \frac{\text{تعداد نودهای خروجی}}{\text{تعداد نودهای ورودی}}$$

کسر اول هر چه به یک نزدیک باشد به معنای مرکزیت بیشتر است ، حال برای آن که از بین فرودگاه هایی که هابیت یکسانی دارند بیشترین اهمیت را تشخیص دهیم آن را در تعداد نود های خروجی فرودگاه ضرب می‌کنیم

که نتایج زیر حاصل شد:

با توجه به نتیجه حاصله SEA فرودگاه هاب تری است.

```
+---------------+---------+--------+----+
|node|input_ord|output_ord|        Hubbility|
+---------------+---------+--------+----+
|SEA|        35|        35|            35.0|
|ANC|        20|        19|           18.05|
|PDX|        16|        16|            16.0|
|JNU|         7|         7|             7.0|
|FAI|         3|         4|        5.333333|
|LAX|         5|         5|             5.0|
```

```
|SFO|        5|        5|             5.0|
|KTN|        4|        4|             4.0|
|AKN|        1|        2|             4.0|
|DEN|        3|        3|             3.0|
|SIT|        3|        3|             3.0|
|PSP|        3|        3|             3.0|
|OAK|        3|        3|             3.0|
+---------------+---------+--------+----+

only showing top 13 rows
```

e: مهمترین ۱۰ فرودگاه را مشخص نموده و درصد اهمیت آنها را مشخص نمایید.

با کمک الگوریتم page rank که خود graphX در اختیار ما قرار میدهد ، مهمترین نقطه را پیدا میکنیم که نتیجه زیر حاصل شد:

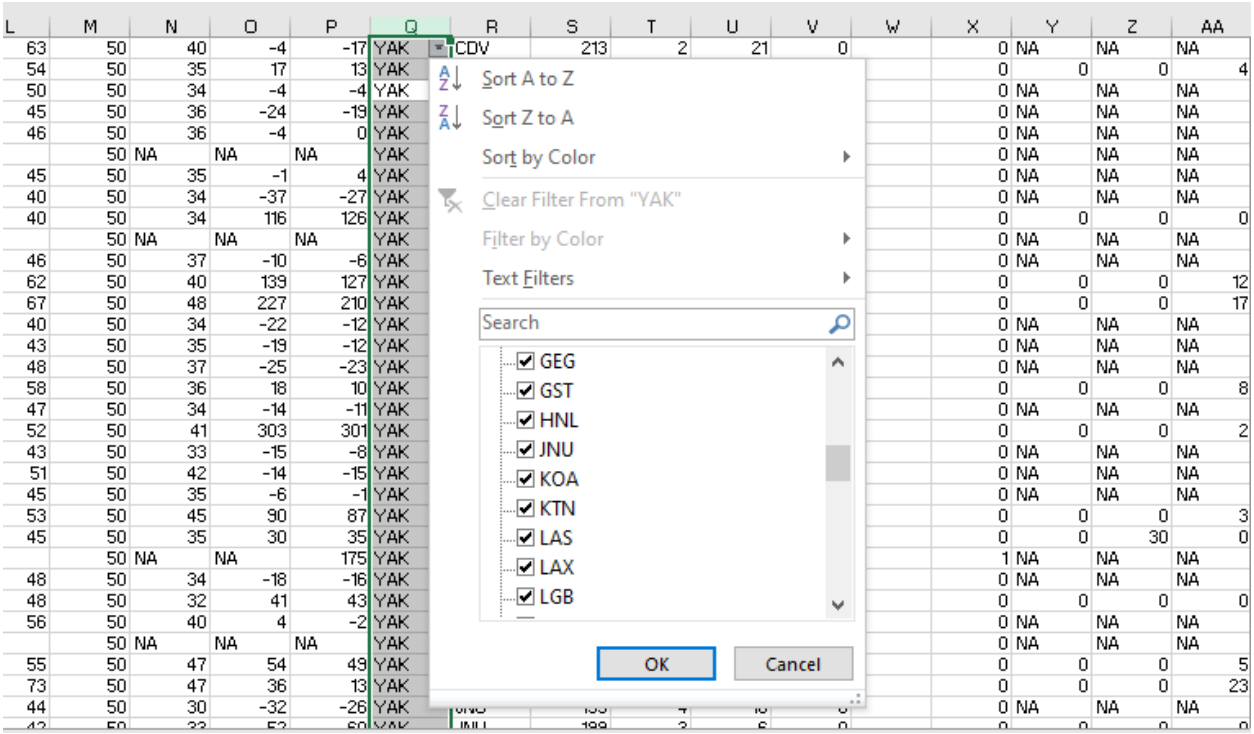
```
+---+----+------------------+
| id|node|              rank|
+---+----+------------------+
| 26| SEA| 8.468267369903197|
| 49| ANC|  4.31196907462454|
| 27| PDX| 3.712331111811354|
| 38| JNU|  1.67211788195928|
| 34| SFO|1.2428303380539998|
| 42| LAX|1.1890815347701815|
|  9| KTN|1.0428605559869997|
| 22| DEN|  0.91843139007883|
| 44| SIT|0.9021335562634846|
|  5| OAK|0.8944569042430728|
+---+----+------------------+
only showing top 10 rows
```

f: ۵ پرواز با بیشترین تاخیر از مبدا فرودگاه JFK به چه فرودگاه هایي هستند و مقدار تاخیر آنها چقدر است.

برای این بخش مجدد دیتافریم جدیدی برای edge ها پارس میکنیم و پردازش مربوطه را انجام می‌دهیم با این تفاوت که بجای معیار distance از مجموع تاخیر های موجود استفاده میکنیم. که به شکل زیر در می آیند: (لازم به ذکر است که ابتدا باید اطلاعات null را فیلتر کنیم.)

```
+---+---+-----+
|src|dis|delay|
+---+---+-----+
| 27| 49|  948|
| 27| 49|  811|
| 27| 49|  672|
+---+---+-----+
only showing top 3 rows
```

با توجه به اینکه در دیتاست Flight فرودگاه JFK وجود ندارد!



محاسبات برای فرودگاه DCA انجام شد :

```
+---+---+-----+
|src|dis|delay|
+---+---+-----+
|  7| 42|  253|
```

```
|  7| 42|  199|
|  7| 26|  197|
|  7| 26|  183|
|  7| 26|  181|
+---+---+-----+
only showing top 5 rows
```

از این فرودگاه با id=7 بیشترین تاخیر ها به مقصد 42 و 26 است که به ترتیب فرودگاه های LAX و SEA هستند. و تاخیر هر کدام نوشته شده.

g: کوتاهترين مسير بين فرودگاه هاي ATL و LAS چقدر است و به ترتيب شامل چه فرودگاه هايي مي باشد؟

با توجه به اینکه ATL در دیتاست داده شده وجود ندارد، فرض کوتاه ترین مسیر ، بین LAS (به عنوان مبدا)و DCA (به عنوان مقصد) حل شد.

برای انجام این بخش از دو روش اتخاذ شد :

۱-استفاده از خود تابع ShortestPaths در کتابخانه graphX که به سادگی تنها یک خروجی دارد و آن نیز فاصله ورتکس ها از هم (و نه distant انها) است.

که با توجه به توضیحات فوق نتیجه به شرح زیر است:

```
branch_distant: Int = 2
```

۲- استفاده از pergel و حرکت کردن روی گراف است ، به این نحوه که یک گراف جدید تعریف کرده و به هر ورتکس آن یک عدد نسبت میدهیم و با استفاده از این متد ، به همسایگی های متفاوت حرکت کرده و تمامی مسیر ها را بررسی میکنیم و کوچک ترین آن را انتخاب میکنیم.

جهت توضیحات بیشتر https://www.youtube.com/watch?v=SYQAOK6JaLE

که نتیجه به فوق به شرح زیر است :

```
Array((7,3195.0))
```

بدین معنا که با فرض شروع از LAS=20 ، فاصله تا DCA=7 ، ۳۱۹۵ مایل است.

در ادامه کد ها قرار گرفت ، همچنین ، کد به صورت یک zeppelin notebook مانند تمرین های قبلی در کنار گزارش قرار گرفت.

```
import org.apache.spark._
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext._
import org.apache.spark.rdd.RDD._
import org.apache.spark.rdd.RDD

//parsing Data

import org.apache.spark.sql.types._
```

```
import org.apache.spark.sql.functions._

val schema = StructType(
    StructField("flightNum", IntegerType, nullable = true) ::
    StructField("tailNum", StringType, nullable = true) ::
    StructField("origin", StringType, nullable = true) ::
    StructField("dest", StringType, nullable = true) ::
    StructField("distance", IntegerType, nullable = true) ::
    StructField("canceled", IntegerType, nullable = true) ::
    StructField("cancellationCode", IntegerType, nullable = true) ::
    StructField("carrierDelay", IntegerType, nullable = true) ::
    StructField("weatherDelay", IntegerType, nullable = true) ::
    StructField("nasDelay", IntegerType, nullable = true) ::
    StructField("securityDelay", IntegerType, nullable = true) ::
    StructField("lateAircraftDelay", IntegerType, nullable = true) ::
    Nil
)

val creditDf = spark.read.format("csv")
  .option("header", value = true)
  .option("delimiter", ",")
  .option("mode", "DROPMALFORMED")
  .schema(schema)
  .load("C:/Users/hosse/Desktop/HW3/flight-m.csv")

  creditDf.show(2)

//definition of graph
import org.apache.spark.graphx._

// create vertices RDD with ID and Name
val vertices=
creditDf.select(
    explode(array("origin","dest")))
    .distinct.rdd.collect
    .map(t=>t(0)).zipWithIndex.map(_.swap)
    .map(t=> (t._1.toLong,t._2.toString))

val vRDD= sc.parallelize(vertices)
```

```
// Now let's define a vertex dataframe because joins are clearer in sparkSQL
val vertexDf = vertices.toList.toDF("id", "node")


// Defining a default vertex called nowhere
val nowhere = "nowhere"

//// Defining edges
val edges=
creditDf
    .join(vertexDf, creditDf("origin") === vertexDf("node"))
    .select('origin,'id as 'orgin_, 'dest,'distance )
    .join(vertexDf, creditDf("dest") === vertexDf("node"))
    .select('orgin_,'id as 'dest_,'distance)
    .distinct
    .rdd.map(row =>
            Edge( row.getAs[Long]("orgin_"), row.getAs[Long]("dest_") ,
row.getAs[Int]("distance")      )
                )
    .collect

val eRDD= sc.parallelize(edges)

// define the graph
val graph = Graph(vRDD,eRDD, nowhere)

val vertixMAP= vertices.toMap
val vertixMAP_inverse=vertices.map(_.swap).toMap

//parta-a
//number of airport:
val number_of_airport=graph.numVertices
vertexDf.show(100)

//parta-b
creditDf
    .join(vertexDf, creditDf("origin") === vertexDf("node"))
    .select('origin,'id as 'orgin_, 'dest,'distance )
    .join(vertexDf, creditDf("dest") === vertexDf("node"))
    .select('orgin_,'id as 'dest_,'distance)
    .distinct
```

```
    .show(180)
val number_of_routes=graph.numEdges

//part b
val
longest_distance=graph.edges.filter(x=>x.attr>1000).collect.toList.toDF("orgin_","dest_
","distance").sort(desc("distance"))
val number_oflongest_distance=longest_distance.count
longest_distance.show(10)

//part b //second view
// print out longest routes
graph.triplets.sortBy(_.attr, ascending=false).map(triplet =>
     "Distance " + triplet.attr.toString + " from " + triplet.srcAttr + " to " +
triplet.dstAttr + ".").take(10).foreach(println)

//part c/output order
val output_ord=graph.triplets.map(t =>
(t.srcAttr,1)).reduceByKey((x,y)=>x+y).collect.toList.toDF("output_node","output_ord").
sort(desc("output_ord"))

output_ord.show(8)

//part c/input order
val input_ord=graph.triplets.map(t =>
(t.dstAttr,1)).reduceByKey((x,y)=>x+y).collect.toList.toDF("input_node","input_ord").so
rt(desc("input_ord"))

input_ord.show(8)

//part c in different way

// Define a reduce operation to compute the highest degree vertex
def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
 if (a._2 > b._2) a else b
}

val maxInDegree: (VertexId, Int) = graph.inDegrees.reduce(max)

val maxOutDegree: (VertexId, Int) = graph.outDegrees.reduce(max)
```

```
val maxDegrees: (VertexId, Int) = graph.degrees.reduce(max)

println("maxInDegree :" +  vertixMAP(maxInDegree._1 ) +"," + maxInDegree._2)
println("maxOutDegree :"+  vertixMAP(maxOutDegree._1) +"," + maxOutDegree._2)
println("maxDegrees :"  +  vertixMAP(maxDegrees._1  ) +"," + maxDegrees._2)

//part D who is HUB
//combine the previous section
val Data =
input_ord.join(output_ord,output_ord("output_node")===input_ord("input_node")).select('
input_node as 'node ,'input_ord,'output_ord)

//Hubbibility = hub ability :))))

val Hub =
Data.withColumn("Hubbility",expr("(output_ord/input_ord)*output_ord")).sort(desc("Hubbi
lity"))

Hub.show(15)

//part e
// use pageRank
val ranks = graph.pageRank(0.1).vertices.collect.toList.toDF("node_rank","rank")
val most_importants_node= ranks.join(vertexDf,ranks("node_rank")===vertexDf("id")
).select("id","node","rank").sort(desc("rank")).show(10)

//part f -creat edge
val new_edges=
creditDf
    .filter(creditDf.col("carrierDelay").isNotNull)
    .filter(creditDf.col("weatherDelay").isNotNull)
    .filter(creditDf.col("nasDelay").isNotNull)
    .filter(creditDf.col("securityDelay").isNotNull)
    .filter(creditDf.col("lateAircraftDelay").isNotNull)
    .join(vertexDf, creditDf("origin") === vertexDf("node"))
    .select('origin,'id as 'orgin_,
'dest,'carrierDelay,'weatherDelay,'nasDelay,'securityDelay,'lateAircraftDelay )
    .join(vertexDf, creditDf("dest") === vertexDf("node"))
    .select('orgin_,'id as
'dest_,'carrierDelay,'weatherDelay,'nasDelay,'securityDelay,'lateAircraftDelay )
    .rdd.map(row =>
```

```
                Edge(row.getAs[Long]("orgin_") , row.getAs[Long]("dest_") ,
row.getAs[Int]("carrierDelay")+

row.getAs[Int]("weatherDelay")+

row.getAs[Int]("nasDelay")+

row.getAs[Int]("securityDelay")+

row.getAs[Int]("lateAircraftDelay")
                                                                        )
            )

    .collect.toList.toDF("src","dis","delay")

new_edges.show(3)

//part f -answering the question
val most_delay_des =
new_edges.filter(new_edges.col("src").isin(vertixMAP_inverse("DCA"))).sort(desc("delay"
)).show(5)

//part g found maximom of branch distant
//1
import org.apache.spark.graphx.lib.ShortestPaths

val Start_node="LAS"
val End_node="DCA"

val branch_distant=ShortestPaths.run(graph, Seq(vertixMAP_inverse(End_node)))
                        .vertices
                        .filter({
                                case(vId, _) =>
                                vId == vertixMAP_inverse(Start_node)
                                })
                        .first._2(vertixMAP_inverse(End_node))

//resource:
//https://www.youtube.com/watch?v=SYQAOK6JaLE
val initialMsg = (Double.PositiveInfinity)//,List("dummy"))
```

```
val initialGraph = graph.mapVertices((id, _) =>
    if (id == vertixMAP_inverse(Start_node) ) 0.0 else Double.PositiveInfinity)

val sssp = initialGraph.pregel(
            initialMsg,
            6,
            EdgeDirection.Out)((id, dist, newDist) =>

                                    math.min(dist, newDist), // Vertex Program

                                    triplet => {  // Send Message
                                                if ( triplet.srcAttr + triplet.attr <
triplet.dstAttr) {
                                                    Iterator((triplet.dstId,
triplet.srcAttr + triplet.attr))
                                                    }
                                                else{
                                                    Iterator.empty
                                                    }
                                                },
                                    (a, b) => math.min(a, b) // Merge Message
                                                )


//sssp.vertices.collect.foreach(println)
sssp.vertices.filter({
                    case(vId, _) =>
                    vId == vertixMAP_inverse(End_node)
                    })
            .collect
```