

Project 3 documentation- Emily Hao, Daojun Liang

Problem1

It is labeled Problem1.scala file.

To run the script, cd into your spark bin directory.

```
$ ./spark-shell -i ~/Desktop/Problem1.scala
```

You should change the above path into the path of the files on your VM.

The path for dataset is built in in the document, so please go into the document and change the path to where your transaction dataset is.

This should open the spark shell and execute the script. Note that T2 and T6 has show commands since you will see T2 T6 results in your terminal as well.

T2 result:

```
T2: org.apache.spark.sql.DataFrame = [TransNumItems: int, sum(TransTotal): double ... 3 more fields]
+-----+-----+-----+-----+-----+
|TransNumItems|sum(TransTotal)|avg(TransTotal)|max(TransTotal)|min(TransTotal)|
+-----+-----+-----+-----+-----+
|1|2.4267489048330688E8|600.2525192024173|999.99915|200.00197|
|6|2.4249778816603088E8|599.8886501665625|999.9993|200.00557|
|3|2.4268549436515808E8|600.3797297638862|999.99866|200.00067|
|5|2.4235919080473328E8|600.1976003029558|999.9986|200.00586|
|9|2.4246856279232788E8|600.0211897855181|999.9991|200.00084|
|4|2.42384000194458E8|599.9871286207257|999.99963|200.00238|
|8|2.4268184401774597E8|600.8880140681849|999.9994|200.00403|
|7|2.4248069013900757E8|600.0809001658275|999.99884|200.00987|
|10|2.4234284782920837E8|600.6777754541674|999.99805|200.00656|
|2|2.4203932090870667E8|599.741114764136|999.99713|200.00072|
+-----+-----+-----+-----+-----+
```

T6 result:

```
T6: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [CustID: int, Count5: bigint ... 1 more field]
+-----+-----+-----+
|CustID|Count5|count|
+-----+-----+-----+
|148|160|70|
|463|255|95|
|471|170|72|
|496|230|87|
|833|135|68|
|1088|190|78|
|1238|190|83|
|1342|260|86|
|1580|150|60|
|1591|215|84|
|1645|225|87|
|1829|265|87|
|1959|170|73|
|2122|195|85|
|2142|220|99|
|2366|180|82|
|2659|240|84|
|2866|190|76|
|3175|180|74|
|3749|185|71|
+-----+-----+-----+
only showing top 20 rows
```

Problem2

Step2:

Java File: ReportCellidTop50

In this task, we have two mapreduce jobs. The second job use the data from the output of first job.

The first job:

Map_first

Input:

datasetPoints.txt

Output:

key: cellId

value: x-axis, y-axis in corresponding cell

Reduce_first

In the reduce phase, we calculate how many points in different cells.

Output:

Key:

NullWritable

Value:

cellId, the number of points in that cell

NOTE: The output file of the first job is in the path :

/Users/daojun/Desktop/mapreduce/output_ReportID/part-r-000000

In step 3, we need to reuse the output file in this job. Because I run it on my personal computer, so in order to run the java code in step 3, you may need to change the path

/Users/daojun/Desktop/mapreduce/output_ReportID/part-r-000000

to HDFS path in step 3. For step 2, you do not need to change the path.

The second job:

Map_second:

Output:

Key:

1

Value:

cellId, the number of points in that cell

By doing so, all the cell id and the corresponding number of points can be read into the reduce job. It can help us calculate Relative-Density index for every cell. Note: we need to consider cell position in order to calculate Relative-Density index for every cell. We need to consider **nine** situations in this case.

Reduce_second:

Key:

Nullwritable

Value:

Top 50 points id and their relative-density index

Below is a screenshot of part of our final result:

| | |
|----|---------------------------|
| 1 | 234073,1.7466307277628033 |
| 2 | 119747,1.7419354838709677 |
| 3 | 192940,1.7360703812316716 |
| 4 | 239618,1.7337461300309598 |
| 5 | 35804,1.7259475218658893 |
| 6 | 167335,1.7191977077363896 |
| 7 | 113351,1.7111111111111111 |
| 8 | 140071,1.706371191135734 |
| 9 | 33589,1.7035040431266846 |
| 10 | 1051,1.702127659574468 |
| 11 | 113181,1.6963350785340314 |
| 12 | 66994,1.6955223880597015 |
| 13 | 150074,1.6904761904761905 |
| 14 | 244766,1.688073394495413 |
| 15 | 224829,1.6878612716763006 |

NOTE: The output file of the first job is in the path :

/Users/daojun/Desktop/mapreduce/output_Report50ID/part-r-00000

In step 3, we need to reuse the output file in this job. Because I run it on my personal computer, so in order to run the java code in step 3, you may need to change the path

/Users/daojun/Desktop/mapreduce/output_Report50ID/part-r-00000

to HDFS path in step 3. For step 2, you do not need to change the path.

Step3:

Java file: ReportNeighborsDensityAndIds.java

In step 3, we only have one mapreduce job, but we need to read two files from step2.

So we use two map tasks to read these two files.

FirstMapper:

Input data path:

"/Users/daojun/Desktop/mapreduce/output_ReportID/part-r-00000"

You need to change the path.

Output:

Key:

1

Value:

"Cells" + every line from the file `"/Users/daojun/Desktop/mapreduce/output_ReportID/part-r-00000"`

SecondMapper:

Input data path:

`"/Users/daojun/Desktop/mapreduce/output_Report50ID/part-r-00000"`

You need to change the path.

Output:

Key:

1

Value:

"Top50" + every line from the file `"/Users/daojun/Desktop/mapreduce/output_Report50ID/part-r-00000"`

Reducer:

Input:

Key:

1

Value:

every line from the file `"/Users/daojun/Desktop/mapreduce/output_ReportID/part-r-00000" +`

`"/Users/daojun/Desktop/mapreduce/output_Report50ID/part-r-00000"`

Output:

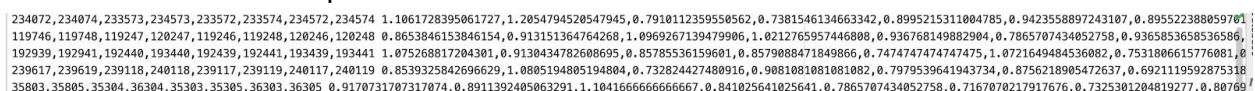
Key:

All the neighborsIDs of the top 50 points

Value:

The corresponding relative-density indexes of these neighbors.

Below is a screenshot of part of the final result:



234072,234074,233573,234573,233572,233574,234574 1.1061728395061727,1.2054794520547945,0.7910112359550562,0.7381546134663342,0.8995215311004785,0.9423558897243107,0.895522380059701
119746,119748,119247,120247,119246,119248,120246,120248 0.8653846153846154,0.913151364764268,1.0969267139479906,1.0212765957446808,0.936768149882904,0.7865707434052758,0.9365853658536586,
192939,192941,192440,193440,192439,192441,193439,193441 1.075268817204301,0.9130434782608695,0.85785536159601,0.8579088471849866,0.7474747474747475,1.0721649484536082,0.7531806615776081,0
239617,239619,239118,240118,239117,239119,240117,240119 0.8539325842696629,1.0805194805194804,0.732824427480916,0.9081081081081082,0.7979539641943734,0.8756218905472637,0.6921119592875318
35803,35805,35304,36304,35303,35305,36303,36305 0.9170731707317074,0.8911392405063291,1.1041666666666667,0.841025641025641,0.7865707434052758,0.7167070217917676,0.7325301204819277,0.80769

Problem3

It is labeled Problem3.scala file.

To run the script, cd into your spark bin directory.

`$. /spark-shell -i ~/Desktop/Problem3.scala`

You should change the above path into the path of the files on your VM. The path for dataset is built in in the document, so please go into the document and change the path to where your transaction dataset is.

This should open the spark shell and execute the script.

I did not have the chance to finish this question because my VM crashed. I was only able to finish writing half way through the for-loop. But I can explain my logic here to demonstrate what my code should be doing.

So outside of the loop, I was able to read the input file, parse the file, initialize the rank table R0. Inside my forloop, I want to join the input link table to rank table to a table [url, rank, url-dest] as shown in the picture. I would also update my rank to [rank/(how many dest link to one url)]. Then I would take the url_dest and rank_sum (rank_sum is the sum of the ranks labeled to each url destination) and use this as my new rank table for the next round. This logic is following the class presentation on Page Rank. I used leftOuterJoin of the two tables to handle the out of edge cases (this is suggested by the Prof).

| url | rank | url-dest |
|-----|---------------|----------|
| a | $\frac{1}{3}$ | b |
| a | $\frac{1}{3}$ | c |
| a | $\frac{1}{3}$ | d |
| e | $\frac{1}{2}$ | b |
| e | $\frac{1}{2}$ | c |

left outer join on url-src

| url-dest | rank_sum |
|----------|---------------|
| b | 5 |
| c | $\frac{5}{6}$ |
| d | $\frac{5}{6}$ |
| | $\frac{1}{3}$ |

group by url-dest
count rank

| url | rank |
|-----|---------------|
| a | |
| b | $\frac{5}{6}$ |
| c | $\frac{5}{6}$ |
| d | $\frac{1}{3}$ |

no - out random -

Sorry that you won't be able to run question 3's script all the way because there are still errors remaining when my VM crashed, so I wasn't able to fix the errors, but the logic should be align with what I have above.