Class – CS6240 Fall-2018 Sec 2
HW – 1
Name- Mustafa Kapadia
Github - https://github.ccs.neu.edu/cs6240f18/mustafa8895/tree/master/HW3

Pseudo-codes

RDD- Page Rank
Create graphRDD
Create ranksRDD

```
For( i = 1 to 10)
        temp =graphRDD.join(ranksRDD)
          .flatmap(
                    If key has no inlinks emit(key, 0) and emit(outlinkNode, pr)
                    Else
                    Emit (outlink, pr)

        temp2 = temp.reduceByKey(_+_)
        delta = value for key 0(dummy node) in temp2 / k*k
        ranksRDD = temp2.map(Add delta to all node pr values except node 0)
        sum = sum of all PRs in ranksRDD except dummy node

return ranksRDD
```

Dataset – Page Rank
Create graphDataSet(from, to)
Create ranksDataset(node, pr)

```
For (i = 1 to 10)

        Temp = graphDataSet.join(ranksDataSet)BY from == node
        StartNodesPR = temp.filter(nodeID % k ==1).map(key=key, pageRank=0)
        OtherNodesPR = temp.groupBy(to).sum
        AllNodesPR = StartNodesPR.union(OtherNodesPR)
        zeroNode = PR of node 0(dummy node) / k*k
        ranksDataSet=Add zeroNode to the pr values of all nodes except the 0 node
        sum = sum of all PRs in ranksDataSet except dummy node

Return ranksDataSet
```

Run Times-

RDD-Page Rank - 8 s
Dataset – Page Rank – 2.4 mins

Actions:

The actions for RDD – lookup(), take()
The actions for DataSet – collect(), show()

PR values after 10 iterations

Node ID	Data Set PR	RDD PR
0	0.010936853	0.01093685
1	1.09E-06	1.09E-06
2	2.17654197831244E-6	2.18E-06
3	3.25E-06	3.25E-06
4	4.31019748102045E-6	4.31E-06
5	5.36E-06	5.36E-06
6	6.40E-06	6.40E-06
7	7.43E-06	7.43E-06
8	8.45E-06	8.45E-06
9	9.46E-06	9.46E-06
10	1.05E-05	1.05E-05
11	1.10E-04	1.10E-04
12	1.10E-04	1.10E-04
13	1.10E-04	1.10E-04
14	1.10E-04	1.10E-04
15	1.10E-04	1.10E-04
16	1.10E-04	1.10E-04
17	1.10E-04	1.10E-04
18	1.10E-04	1.10E-04
19	1.10E-04	1.10E-04
20	1.10E-04	1.10E-04
21	1.10E-04	1.10E-04
22	1.10E-04	1.10E-04
23	1.10E-04	1.10E-04

24	1.10E-04	1.10E-04
25	1.10E-04	1.10E-04
26	1.10E-04	1.10E-04
27	1.10E-04	1.10E-04
28	1.10E-04	1.10E-04
29	1.10E-04	1.10E-04
30	1.10E-04	1.10E-04
31	1.10E-04	1.10E-04
32	1.10E-04	1.10E-04
33	1.10E-04	1.10E-04
34	1.10E-04	1.10E-04
35	1.10E-04	1.10E-04
36	1.10E-04	1.10E-04
37	1.10E-04	1.10E-04
38	1.10E-04	1.10E-04
39	1.10E-04	1.10E-04
40	1.10E-04	1.10E-04
41	1.10E-04	1.10E-04
42	1.10E-04	1.10E-04
43	1.10E-04	1.10E-04
44	1.10E-04	1.10E-04
45	1.10E-04	1.10E-04
46	1.10E-04	1.10E-04
47	1.10E-04	1.10E-04
48	1.10E-04	1.10E-04
49	1.10E-04	1.10E-04
50	1.10E-04	1.10E-04
51	1.10E-04	1.10E-04
52	1.10E-04	1.10E-04
53	1.10E-04	1.10E-04
54	1.10E-04	1.10E-04
55	1.10E-04	1.10E-04
56	1.10E-04	1.10E-04
57	1.10E-04	1.10E-04
58	1.10E-04	1.10E-04
59	1.10E-04	1.10E-04
60	1.10E-04	1.10E-04
61	1.10E-04	1.10E-04
62	1.10E-04	1.10E-04

63	1.10E-04	1.10E-04
64	1.10E-04	1.10E-04
65	1.10E-04	1.10E-04
66	1.10E-04	1.10E-04
67	1.10E-04	1.10E-04
68	1.10E-04	1.10E-04
69	1.10E-04	1.10E-04
70	1.10E-04	1.10E-04
71	1.10E-04	1.10E-04
72	1.10E-04	1.10E-04
73	1.10E-04	1.10E-04
74	1.10E-04	1.10E-04
75	1.10E-04	1.10E-04
76	1.10E-04	1.10E-04
77	1.10E-04	1.10E-04
78	1.10E-04	1.10E-04
79	1.10E-04	1.10E-04
80	1.10E-04	1.10E-04
81	1.10E-04	1.10E-04
82	1.10E-04	1.10E-04
83	1.10E-04	1.10E-04
84	1.10E-04	1.10E-04
85	1.10E-04	1.10E-04
86	1.10E-04	1.10E-04
87	1.10E-04	1.10E-04
88	1.10E-04	1.10E-04
89	1.10E-04	1.10E-04
90	1.10E-04	1.10E-04
91	1.10E-04	1.10E-04
92	1.10E-04	1.10E-04
93	1.10E-04	1.10E-04
94	1.10E-04	1.10E-04
95	1.10E-04	1.10E-04
96	1.10E-04	1.10E-04
97	1.10E-04	1.10E-04
98	1.10E-04	1.10E-04
99	1.10E-04	1.10E-04
100	1.10E-04	1.10E-04

Page Rank sum for each iteration

Iteration	Data Set Sum(PR)	RDD Sum(PR)
1	1.0000000020000000	0.9999999999999580
2	0.9999999999999990	1.0000000000000600
3	1.0000000000000000	1.0000000000000500
4	1.0000000000000000	1.0000000000000000
5	1.0000000000000000	0.9999999999999550
6	0.9999999999999990	0.9999999999999620
7	1.0000000000000000	1.0000000000000200
8	1.0000000000000000	1.0000000000000300
9	0.9999999999999990	1.0000000000000600
10	1.0000000000000000	1.0000000000000200

Number of Joins:

Dataset:

- The dataset implementation without checkpointing has as many joins in an iteration as the iteration number. The joins from previous iterations are recomputed for the action. This leads to a total of 55 joins for 10 iterations.

- However, checkpointing solves this problem by storing the already computed joins in a temporary folder and using it to perform the join in the next iteration. This reduces the number of joins to 1 per iteration i.e 10 in total.

RDD:

- The toDebugString() command retrieves the RDD lineage from which it can be observed that only 1 join is performed per iteration i.e 10 joins in total.
- It creates map partitions for each of the RDDs it joins before joining them as seen below

```
(4) MapPartitionsRDD[61] at map at twitterFollowers.scala:72 []
 |  ShuffledRDD[60] at reduceByKey at twitterFollowers.scala:66 []
 +-(4) MapPartitionsRDD[59] at flatMap at twitterFollowers.scala:60 []
    |  MapPartitionsRDD[58] at join at twitterFollowers.scala:59 []
    |  MapPartitionsRDD[57] at join at twitterFollowers.scala:59 []
    |  CoGroupedRDD[56] at join at twitterFollowers.scala:59 []
```

MR – Diameter Pseudo Code

Job 1: Creates adjacency list and marks source nodes as active

```
Map(from, to)
    Emit(from, to)
    Emit(to, 0)

Reduce(from, to)
    List adjList = []
    For value in values:
        If value != 0 // dummy
            adjList.add(value)

    Create Node with adjacency list
    If node Is a source node
        Node.isActive = true

    Emit Node
```

Job 2: finds distance of nodes from each source

```
Map(Node)

    If node is a source
        Set node.distance from self to 0

    Emit node Obj

    If node.isActive:
        For each adjnode in adjacency list of this node:
            For each source:
                Update adjnode's distance from source to this node's distance + 1
            Emit(AdjNode)


Reduce

    dminMap // stores min distance from source so far, initialized to
        infinity

    Node m = null
    For node in values:
        If node.isObj // node object found
            M = node
```

```
            M.isActive = false

    Else // adjacency object found
        For each source s
            If node.distance(s) < dminMap(s)
                dminMap(s) = node.distance(s)


    For each source s
        If dminMap(s) < m.distance(s)
            m.distance(s) = dminMap(s)
            m.setActive()
            Increment counter

    Emit(m)
```

Job 3 : Finds max of minimum distances

```
Map(node)
For each source s
    Emit(s, node.distance(s))

Reduce(source, Distances)
    Emit(source, max(Distances))
```

Spark-RDD Shortest Path Pseudo Code

```
    // Create adjacency list from input file

    adjList = Input.map(split(“,”)).reduceByKey((a,b) => List.concat(a,b))

    persist adjList

    distances = mapSources(adjList) // if source, sets distance to itself as 0, else
                                        infinity


    for (iterationCount <- 1 to k)
    distances= adjlist.join( distances ) // use common partitioner
    .flatMap( (n, adjList, currentPRofN) => extractVertices(adjList, (ds1, ds2))
    .reduceByKey( (x, y) => (x + y) )} // for s1 and s2
```

Maxd(distances)

Return max of minimum distances from each source in distances

extractVertices

returns each vertex id m in n's adjacency list as (m, (n's distance from s1 +1, n's distance from s2+1))

Spark-Dataset Diameter

Create Graph(to, from)

Persist Graph

Create Distance (if node is source, set distance to itself as 0, else infinity)

For I = 1 to 10

Graph.join(Distance)

.union(Distance) // preserves nodes with no inlinks

.groupBy(node)

.min(sd1, sd2) // sd1: Distance from s1, sd2: Distance from s2

Maxd(distances)

Return max of minimum distances from each source in distances

---

Spark/MR	Number of Machines	K	Run Time	Longest Shortest Path
MR	6	5	32 minutes	9
MR	11	5	22 minutes	9
Spark - RDD	6	2	2 hours, 21 minutes	9
Spark - RDD	11	2	2 hours, 12 minutes	9
Spark - Dataset	6	2	30 minutes	9
Spark - Dataset	11	2	20 minutes	9

References:

https://stackoverflow.com/questions/27002161/reduce-a-key-value-pair-into-a-key-list-pair-with-apache-spark - Creating adjacency list

https://www.safaribooksonline.com/library/view/high-performance-spark/9781491943199/ch04.html - partitioner

https://google.github.io/gson/apidocs/com/google/gson/Gson.html - Gson