1. Pseudo-code for PageRank Program in Spark Scala

- The following is the Pseudo-code which heavily borrowed Scala syntax.

```
/* PAGERANK-SPARK-SCALA */

val k ← user set k value (number of linear chains)

val outlinkListBuffer ← List Buffer of tuple (Int, Int)
val rankListBuffer = ← list buffer of tuple (Int, Double) containing dummy page (0, 0)

// generate outlink (p1 → p2) values
for (i ← 1 to pow(k, 2)) {
  if (i % k == 0) {
    outlinkListBuffer += ((i, 0))
  } else {
    outlinkListBuffer += ((i, i + 1))
  }
}

// generate initial PageRank values
for (i ← 1 to pow(k, 2).toInt) {
  rankListBuffer += ((i, 1 / pow(k, 2)))
}

// create outlink RDD
val outlinkRDD = sc.parallelize(outlinkListBuffer).partitionBy(new HashPartitioner(10))

// create inlink RDD and join with outlink RDD
val inlinkRDD = outlinkRDD
                .map{case (from, to) ⇒ (to, from)}
                .partitionBy(outlinkRDD.partitioner.get)
                // outer-join with outlink graph to get inlink/outlink of each page
                .fullOuterJoin(outlinkRDD)
                // map in appropriate format to help join with PageRank graph later on
                .map {
                  case(page, (inlink, outlink)) ⇒ {
                    if (page == 0) {
                      (inlink.getOrElse(0), page) // handle dummy page
                    } else {
                      (inlink.getOrElse(-1), page) // handle other every page
                    }
                  }
                }
                .persist()

// create PageRank RDD
var rankRDD = sc.parallelize(rankListBuffer).partitionBy(outlinkRDD.partitioner.get)

val alpha = 0.85

// iteration until convergence
val iteration ← user set number of PageRank iterations
for (i ← 1 to iteration) {

  // join edge RDD with PageRank RDD and output according to the page type
  val joinRDD = inlinkRDD
                .leftOuterJoin(rankRDD)
                .map {
                  case(outlink, (page, someRank)) ⇒ {

                    val pageRank = (1 - alpha) / pow(k, 2)

                    // if the page has no inlink, the value is set to 0 by default
                    val rank = someRank.getOrElse(0.toDouble)

                    if (rank == 0.toDouble) {
                      (page, pageRank)  // handle page with no inlink (only outgoing link)
                    } else if (page == 0) {
                      (page, rank)  // handle dangling page
                    } else {
                      (page, pageRank + alpha * rank) // handle page with outlink
                    }
                  }
                }
                .reduceByKey(_ + _) // aggregate to get dangling page PR mass

  // read out total dangling PageRank mass in dummy page 0
  val danglingPageRankSum = joinRDD.lookup(0).head

  // distribute dangling PageRank mass evenly over all real pages
  rankRDD = joinRDD
            .map {
              case(page, rank) ⇒ {
                if (page == 0) {
                  (page, 0.toDouble)  // if page is dummy, set its PageRank to 0
                } else {
                  (page, rank + alpha * danglingPageRankSum / pow(k, 2))  // update PageRank
                }
              }
            }

}

// for debugging purpose, check if PageRank values sum up to 1
print ← sum up all the PageRank values in RankRDD

// group to one output file, sort, and print first 20 records (page 0 ~ 19)
print ← rankRDD.repartition(1).sortByKey().take(20)
```

2. Source code for PageRank Program in Spark Scala

- https://github.com/CS6240/hw-4-spark-jill666666/blob/master/src/main/scala/pr/PageRank.scala

3. PageRank Program Output for k=100 and iterations=10

```
(0,0.0)
(1,1.5888125735610845E-5)
(2,2.9391220523344354E-5)
(3,4.086705694684288E-5)
(4,5.061974152456419E-5)
(5,5.890776462131797E-5)
(6,6.595084287305438E-5)
(7,7.193573524791716E-5)
(8,7.702118669812162E-5)
(9,8.134213026403417E-5)
(10,8.501325886262295E-5)
(11,1.047006992966952E-4)
(12,1.047006992966952E-4)
(13,1.047006992966952E-4)
(14,1.047006992966952E-4)
(15,1.047006992966952E-4)
(16,1.047006992966952E-4)
(17,1.047006992966952E-4)
(18,1.047006992966952E-4)
(19,1.047006992966952E-4)
```

- For each tuple (p1, p2), p1 is a page, and p2 is its corresponding PageRank.

4. PageRank RDD Lineage Report

- Iteration 1

```
2021-11-05 22:58:38,698 INFO root: (4) MapPartitionsRDD[12] at map at PageRank.scala:103 []
 |  ShuffledRDD[11] at reduceByKey at PageRank.scala:96 []
 +-(4) MapPartitionsRDD[10] at map at PageRank.scala:79 []
    |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:78 []
    |  MapPartitionsRDD[8] at leftOuterJoin at PageRank.scala:78 []
    |  CoGroupedRDD[7] at leftOuterJoin at PageRank.scala:78 []
    +-(4) MapPartitionsRDD[5] at map at PageRank.scala:55 []
       |  |  MapPartitionsRDD[4] at fullOuterJoin at PageRank.scala:53 []
       |  |  MapPartitionsRDD[3] at fullOuterJoin at PageRank.scala:53 []
       |  |  CoGroupedRDD[2] at fullOuterJoin at PageRank.scala:53 []
       |  +-(4) MapPartitionsRDD[1] at map at PageRank.scala:51 []
       |  |  |  ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
       |  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
       +-(4) ParallelCollectionRDD[6] at parallelize at PageRank.scala:66 []
```

- Iteration 2

```
2021-11-05 22:57:42,685 INFO root: (4) MapPartitionsRDD[18] at map at PageRank.scala:103 []
 |  ShuffledRDD[17] at reduceByKey at PageRank.scala:96 []
 +-(4) MapPartitionsRDD[16] at map at PageRank.scala:79 []
    |  MapPartitionsRDD[15] at leftOuterJoin at PageRank.scala:78 []
    |  MapPartitionsRDD[14] at leftOuterJoin at PageRank.scala:78 []
    |  CoGroupedRDD[13] at leftOuterJoin at PageRank.scala:78 []
    +-(4) MapPartitionsRDD[5] at map at PageRank.scala:55 []
       |  |  MapPartitionsRDD[4] at fullOuterJoin at PageRank.scala:53 []
       |  |  MapPartitionsRDD[3] at fullOuterJoin at PageRank.scala:53 []
       |  |  CoGroupedRDD[2] at fullOuterJoin at PageRank.scala:53 []
       |  +-(4) MapPartitionsRDD[1] at map at PageRank.scala:51 []
       |  |  |  ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
       |  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
       +-(4) MapPartitionsRDD[12] at map at PageRank.scala:103 []
          |  ShuffledRDD[11] at reduceByKey at PageRank.scala:96 []
          +-(4) MapPartitionsRDD[10] at map at PageRank.scala:79 []
             |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:78 []
             |  MapPartitionsRDD[8] at leftOuterJoin at PageRank.scala:78 []
             |  CoGroupedRDD[7] at leftOuterJoin at PageRank.scala:78 []
             +-(4) MapPartitionsRDD[5] at map at PageRank.scala:55 []
                |  |  MapPartitionsRDD[4] at fullOuterJoin at PageRank.scala:53 []
                |  |  MapPartitionsRDD[3] at fullOuterJoin at PageRank.scala:53 []
                |  |  CoGroupedRDD[2] at fullOuterJoin at PageRank.scala:53 []
                |  +-(4) MapPartitionsRDD[1] at map at PageRank.scala:51 []
                |  |  |  ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
                |  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
                +-(4) ParallelCollectionRDD[6] at parallelize at PageRank.scala:66 []
```

- Iteration 3

```
2021-11-05 22:53:03,644 INFO root: (4) MapPartitionsRDD[24] at map at PageRank.scala:103 []
 |  ShuffledRDD[23] at reduceByKey at PageRank.scala:96 []
 +-(4) MapPartitionsRDD[22] at map at PageRank.scala:79 []
    |  MapPartitionsRDD[21] at leftOuterJoin at PageRank.scala:78 []
    |  MapPartitionsRDD[20] at leftOuterJoin at PageRank.scala:78 []
    |  CoGroupedRDD[19] at leftOuterJoin at PageRank.scala:78 []
    +-(4) MapPartitionsRDD[5] at map at PageRank.scala:55 []
    |  |  MapPartitionsRDD[4] at fullOuterJoin at PageRank.scala:53 []
    |  |  MapPartitionsRDD[3] at fullOuterJoin at PageRank.scala:53 []
    |  |  CoGroupedRDD[2] at fullOuterJoin at PageRank.scala:53 []
    |  +-(4) MapPartitionsRDD[1] at map at PageRank.scala:51 []
    |  |  |  ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
    |  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
    +-(4) MapPartitionsRDD[18] at map at PageRank.scala:103 []
       |  ShuffledRDD[17] at reduceByKey at PageRank.scala:96 []
       +-(4) MapPartitionsRDD[16] at map at PageRank.scala:79 []
          |  MapPartitionsRDD[15] at leftOuterJoin at PageRank.scala:78 []
          |  MapPartitionsRDD[14] at leftOuterJoin at PageRank.scala:78 []
          |  CoGroupedRDD[13] at leftOuterJoin at PageRank.scala:78 []
          +-(4) MapPartitionsRDD[5] at map at PageRank.scala:55 []
          |  |  MapPartitionsRDD[4] at fullOuterJoin at PageRank.scala:53 []
          |  |  MapPartitionsRDD[3] at fullOuterJoin at PageRank.scala:53 []
          |  |  CoGroupedRDD[2] at fullOuterJoin at PageRank.scala:53 []
          |  +-(4) MapPartitionsRDD[1] at map at PageRank.scala:51 []
          |  |  |  ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
          |  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
          +-(4) MapPartitionsRDD[12] at map at PageRank.scala:103 []
             |  ShuffledRDD[11] at reduceByKey at PageRank.scala:96 []
             +-(4) MapPartitionsRDD[10] at map at PageRank.scala:79 []
                |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:78 []
                |  MapPartitionsRDD[8] at leftOuterJoin at PageRank.scala:78 []
                |  CoGroupedRDD[7] at leftOuterJoin at PageRank.scala:78 []
                +-(4) MapPartitionsRDD[5] at map at PageRank.scala:55 []
                |  |  MapPartitionsRDD[4] at fullOuterJoin at PageRank.scala:53 []
                |  |  MapPartitionsRDD[3] at fullOuterJoin at PageRank.scala:53 []
                |  |  CoGroupedRDD[2] at fullOuterJoin at PageRank.scala:53 []
                |  +-(4) MapPartitionsRDD[1] at map at PageRank.scala:51 []
                |  |  |  ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
                |  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
                +-(4) ParallelCollectionRDD[6] at parallelize at PageRank.scala:66 []
```

Details for Job 0

Status: SUCCEEDED
Completed Stages: 6

▸ Event Timeline
▸ DAG Visualization

▾ **Completed Stages (6)**

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	lookup at PageRank.scala:99 +details	2021/11/06 03:06:55	0.2 s	1/1			19.7 KiB	
4	map at PageRank.scala:79 +details	2021/11/06 03:06:53	2 s	2/2			133.6 KiB	39.5 KiB
3	map at PageRank.scala:55 +details	2021/11/06 03:06:52	0.9 s	2/2			158.3 KiB	81.8 KiB
2	parallelize at PageRank.scala:66 +details	2021/11/06 03:06:46	0.8 s	2/2				51.8 KiB
1	map at PageRank.scala:51 +details	2021/11/06 03:06:46	1.0 s	2/2				79.1 KiB
0	parallelize at PageRank.scala:47 +details	2021/11/06 03:06:46	0.9 s	2/2				79.2 KiB

- If we look at the details for Job 0, it parallelizes the Ranks RDD which is the n followed by map and lookup operations.

Details for Job 1

Status: SUCCEEDED
Completed Stages: 4
Skipped Stages: 5

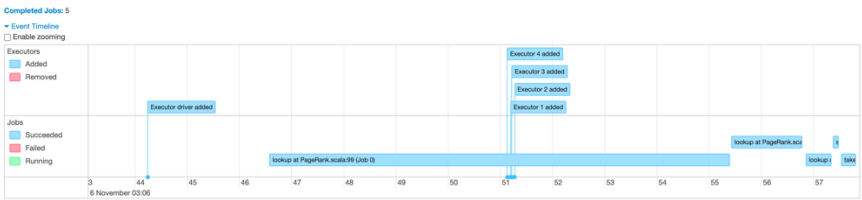
▸ Event Timeline
▸ DAG Visualization

▾ **Completed Stages (4)**

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	lookup at PageRank.scala:99 +details	2021/11/06 03:06:56	0.2 s	1/1			19.8 KiB	
13	map at PageRank.scala:79 +details	2021/11/06 03:06:55	0.8 s	2/2			135.1 KiB	39.6 KiB
12	map at PageRank.scala:103 +details	2021/11/06 03:06:55	0.1 s	2/2			39.5 KiB	53.3 KiB
11	map at PageRank.scala:55 +details	2021/11/06 03:06:55	0.4 s	2/2			158.3 KiB	81.8 KiB

- Job 1 and Job 2 is almost identical in terms of the operations. We can also ch eck skipped stages since they have already been computed in the previous ro und.



- Event timeline shows how each job is being triggered. The jobs include joining the Ranks RDD with the graph RDD, mapping to get the PageRank mass and calculate total dangling pages PageRank values.

- And we have Job 3 which sums up the PageRank values of all pages. This is for the debugging purpose to check whether the result sums up to 1. Finally, we have Job 4 that 'take' first 20 records (page 0 ~ 19) which is our final ou tput.

5. Spark Cache and Reusability

a. Below is the lineage of Ranks RDD after 3 iterations.

```
2021-11-05 21:59:58,537 INFO root: (4) MapPartitionsRDD[24] at map at PageRank.scala:103 []
 |  ShuffledRDD[23] at reduceByKey at PageRank.scala:96 []
 +-(4) MapPartitionsRDD[22] at map at PageRank.scala:79 []
    |  MapPartitionsRDD[21] at leftOuterJoin at PageRank.scala:78 []
    |  MapPartitionsRDD[20] at leftOuterJoin at PageRank.scala:78 []
    |  CoGroupedRDD[19] at leftOuterJoin at PageRank.scala:78 []
    +-(4) MapPartitionsRDD[5] at map at PageRank.scala:55 []
    |  |  MapPartitionsRDD[4] at fullOuterJoin at PageRank.scala:53 []
    |  |  MapPartitionsRDD[3] at fullOuterJoin at PageRank.scala:53 []
    |  |  CoGroupedRDD[2] at fullOuterJoin at PageRank.scala:53 []
    |  +-(4) MapPartitionsRDD[1] at map at PageRank.scala:51 []
    |  |  |  ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
    |  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
    +-(4) MapPartitionsRDD[18] at map at PageRank.scala:103 []
       |  ShuffledRDD[17] at reduceByKey at PageRank.scala:96 []
       +-(4) MapPartitionsRDD[16] at map at PageRank.scala:79 []
          |  MapPartitionsRDD[15] at leftOuterJoin at PageRank.scala:78 []
          |  MapPartitionsRDD[14] at leftOuterJoin at PageRank.scala:78 []
          |  CoGroupedRDD[13] at leftOuterJoin at PageRank.scala:78 []
          +-(4) MapPartitionsRDD[5] at map at PageRank.scala:55 []
          |  |  MapPartitionsRDD[4] at fullOuterJoin at PageRank.scala:53 []
          |  |  MapPartitionsRDD[3] at fullOuterJoin at PageRank.scala:53 []
          |  |  CoGroupedRDD[2] at fullOuterJoin at PageRank.scala:53 []
          |  +-(4) MapPartitionsRDD[1] at map at PageRank.scala:51 []
          |  |  |  ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
          |  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
          +-(4) MapPartitionsRDD[12] at map at PageRank.scala:103 []
             |  ShuffledRDD[11] at reduceByKey at PageRank.scala:96 []
             +-(4) MapPartitionsRDD[10] at map at PageRank.scala:79 []
                |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:78 []
                |  MapPartitionsRDD[8] at leftOuterJoin at PageRank.scala:78 []
                |  CoGroupedRDD[7] at leftOuterJoin at PageRank.scala:78 []
                +-(4) MapPartitionsRDD[5] at map at PageRank.scala:55 []
                |  |  MapPartitionsRDD[4] at fullOuterJoin at PageRank.scala:53 []
                |  |  MapPartitionsRDD[3] at fullOuterJoin at PageRank.scala:53 []
                |  |  CoGroupedRDD[2] at fullOuterJoin at PageRank.scala:53 []
                |  +-(4) MapPartitionsRDD[1] at map at PageRank.scala:51 []
                |  |  |  ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
                |  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:47 []
                +-(4) ParallelCollectionRDD[6] at parallelize at PageRank.scala:66 []
```

Adding the print statement didn't change the lineage, and each of the iteration pri nt statement has been printed only once (e.g., 1, 2, 3, …). We can also find out t hat Spark is smart enough to re-use the pre-computed results.

b. We can certainly see from the log how Spark manages the jobs efficiently by per forming the lookup actions. Below is part of lookup logs copied from the log file.

- 2021-11-05 21:55:16,253 INFO spark.SparkContext: Starting job: lookup at PageRank.scala:99
- 2021-11-05 21:55:16,583 INFO scheduler.DAGScheduler: Got job 0 (lookup at PageRank.scala:99) with 1 output partitions
- 2021-11-05 21:55:16,583 INFO scheduler.DAGScheduler: Final stage: ResultStage 5 (lookup at PageRank.scala:99)
- 2021-11-05 21:55:18,001 INFO scheduler.DAGScheduler: ResultStage 5 (lookup at PageRank.scala:99) finished in 0.075 s
- 2021-11-05 21:55:18,006 INFO scheduler.DAGScheduler: Job 0 finished: lookup at PageRank.scala:99, took 1.752810 s
- 2021-11-05 21:55:18,024 INFO spark.SparkContext: Starting job: lookup at PageRank.scala:99
- 2021-11-05 21:55:18,029 INFO scheduler.DAGScheduler: Got job 1 (lookup at PageRank.scala:99) with 1 output partitions
- 2021-11-05 21:55:18,029 INFO scheduler.DAGScheduler: Final stage: ResultStage 14 (lookup at PageRank.scala:99)

…

- 2021-11-05 21:55:19,402 INFO scheduler.DAGScheduler: Job 8 finished: lookup at PageRank.scala:99, took 0.118823 s
- 2021-11-05 21:55:19,418 INFO scheduler.DAGScheduler: Got job 9 (lookup at PageRank.scala:99) with 1 output partitions
- 2021-11-05 21:55:19,418 INFO scheduler.DAGScheduler: Final stage: ResultStage 194 (lookup at PageRank.scala:99)
- 2021-11-05 21:55:19,525 INFO scheduler.DAGScheduler: ResultStage 194 (lookup at PageRank.scala:99) finished in 0.012 s
- 2021-11-05 21:55:19,525 INFO scheduler.DAGScheduler: Job 9 finished: lookup at PageRank.scala:99, took 0.110396 s

As we can see from the logs, even without Cache() or Persist() operations, Sp ark re-uses Ranks RDD result that has been pre-computed in the previous ite ration.

c. Refer to the link to the log files created after running two programs, one without Cache(), and the other one using Cache().

- Link: https://github.com/CS6240/hw-4-spark-jill666666/tree/master/logs

One most prominent difference between two cases is that the caching makes Spark program finds the blocks and store the blocks as values in memory to re-use them.

Below is some of the logs that only appears when cached.

- 2021-11-05 22:22:58,946 INFO storage.BlockManager: Found block rdd_66_2 locally
- 2021-11-05 22:22:58,946 INFO storage.BlockManager: Found block rdd_66_0 locally
- 2021-11-05 22:22:58,946 INFO storage.BlockManager: Found block rdd_66_3 locally
- 2021-11-05 22:22:58,946 INFO storage.BlockManager: Found block rdd_66_1 locall

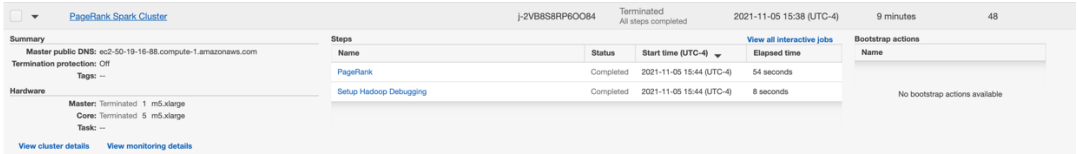
Even though program without Cache() stores blocks such as broadcast_8 as values in memory, following are the logs that are only distinct to the program with Cache().

- 2021-11-05 22:22:57,028 INFO memory.MemoryStore: Block rdd_12_0 stored as values in memory (estimated size 87.9 KiB, free 366.2 MiB)
- 2021-11-05 22:22:57,029 INFO memory.MemoryStore: Block rdd_12_1 stored as values in memory (estimated size 87.9 KiB, free 366.1 MiB)

One another interesting thing is that we can see the CachedPartitions in the lineage of the Ranks RDD.

6. Spark Scala PageRank Program on EMR (1 Master & 5 Workers)

- k = 10,000 caused java heap space 'OutOfMemory' error. Setting k = 1,000 instead with 10 iterations resulted in the following output.
- Output & Log: https://github.com/CS6240/hw-4-spark-jill666666/tree/master/aws-output
- Running Time: Approximately 1 minute



- Thoughts: There are some approaches I would like to take, if more time has been allowed, to address the memory problem. One thing can be generating the input in parallel and create input RDD that can save a way much more memory. Another solution can be storing the intermediate output every iteration in the file system, like what we did in the MapReduce PageRank program.

7. Pseudo-code for PageRank Program in MapReduce

```
/* PAGERANK-MAPREDUCE */


input <- each line of the text follows the format (page, [outlinks], PageRank)

PR_MASS_COUNTER <- global counter that sums up the PageRank mass values
TOTAL_NUM_VERTICES <- global counter that counts the total number of vertices

// takes in the input text file and calculate the total number of vertices
// and the dangling page PageRank mass sum.
DanglingPageMapper(text) {
  tokens <- split input text file by "\n"
  for (String token in tokens) {
    adjacencyList, pageRank <- unpack tokens
    if (adjacencyList is empty) {
      PR_MASS_COUNTER <- increment by PageRank value
    }
    TOTAL_NUM_VERTICES <- increment by 1
  }
}

// also takes in the input text file and emit pair (vertex ID, Vertex object).
// each Vertex object contains its ID, pageRank, adjacencyList, and boolean
// value flag (dangling page ? true : else false).
// emit twice, one with key vertex ID, and the another one with key inlink ID.
PageRankMapper(text) {
  tokens <- split input text file by "\n"
  for (String token in tokens) {
    vertexId, adjacencyList, pageRank <- unpack tokens
    if (adjacencyList is empty) {
      inlinkId <- get inlink vertex from adjacencyList
      emit(inlinkId, Vertex(vertexId, pageRank, adjacencyList, flag false))
    }
    emit(vertexId, Vertex(vertexId, pageRank, adjacencyList, flag true))
  }
}
```

```
PageRankReducer(vertexId, [vertices ...]) {

  // retrieve value from the global counters which will help calculate and
  // update the new PageRank for each page
  Setup() {
    danglingPageRankSum <- get value from global counter PR_MASS_COUNTER
    totalNumVertices <- get value from global counter TOTAL_NUM_VERTICES
  }

  alpha = 0.85
  adjacencyList = []
  newPageRank = ((1 - alpha) / totalNumVertices)
              + (alpha * danglingPageRankSum / totalNumVertices)

  // iterate through object vertices
  // for each vertex, if flag shows false (not a dangling page), calculate the
  // final PageRank.
  // get the page's true adjacency list.
  for (object vertex in vertices) {
    if (vertex.flag is false) {
      newPageRank += alpha * vertex.pageRank / vertex.adjacencyList.length
    } else {
      adjacencyList <- vertex.adjacencyList
    }
  }

  // here the reducer output has same format as the input, since this output
  // should become the input for the mapper in next iteration
  emit(vertexId, (adjacencyList, newPageRank))
}


// chain multiple jobs and iteratively run PageRank program
RunMapReduce() {
  numIterations <- number of iterations
  for (iter from 1 to numIterations) {
    job1 <- assign job DanglingPageMapper only
    job2 <- assign job PageRankMapper followed by PageRankReducer
    // takes in the input and run job1
    input -> run job1
    // run job2 and store the result in output
    output <- run job2
    // the output becomes the new input and will be read in the next iteration
    input <- output
  }
}
```

8. Source code for PageRank Program in MapReduce

- https://github.com/CS6240/hw-4-mapreduce-jill666666/blob/master/src/main/java/pr/PageRank.java

9. MapReduce PageRank Program on EMR (1 Master & 5 Workers)

- Logs: https://github.com/CS6240/hw-4-mapreduce-jill666666/tree/master/aws-log
- Output: https://github.com/CS6240/hw-4-mapreduce-jill666666/tree/master/aws-output
- Running Time: Approximately 50 minutes