CS 6240 HW4
Derek Grubis

Github Link (and README): https://github.ccs.neu.edu/cs6240-f19/dgrubis-Assignment-4
Disclaimer: Due to unfortunate circumstances, this homework is incomplete. MapReduce more so than Spark in terms of code. My apologies. I still tried to portray my understanding of the homework without executable code.

PageRank in Spark

```
generate val graph
val graphDangling = graph.flatMap{case(node, adjList) => if (adjList == None) "0" else
adjList}

var PR = ids.mapValues(v => if (v!= 0) (1.0 / pow(k,2)) else 0)


for (i <- 1 to max_iter) {
  val contr = ids.join(PR).values.flatMap{case (nodes, pr) =>
    val size = nodes.size
    nodes.map(node => (node, pr / size))
   }
   var danglingPR = PR.lookup("0")(0)
   PR = contr.reduceByKey(_+_).mapValues(0.15 + 0.85 * _)
   PR.toDebugString
  }

val results = PR.collect()
results.foreach(tup => println(s"${tup._1} has rank:  ${tup._2} ."))
```

The only operations that perform an action are collect and foreach. Join, mapValues, flatValues, reduceByKey are all transformations.

Looking at the lineage at each iterations shows whether an action was actually performed or if the RDD was transformed at each lineage.

Spark is not smart enough to figure out if it can reuse RDDs computed for an earlier action. It recomputes them at every action however using cache() or persist() hints for the Spark optimizer to not re-compute the RDD again from a previous action.

PageRank in MapReduce

```
map(id n, vertex N) {

        for i in N.adjacencyList {
```

```
        if N.adjacencyList[i] == 0 {
                add vertex 0 to N.adjacencyList[i]
}
        else {
                nothing
}
}
        add 0 to N
        add all vertices with outgoing links to 0’s adjacencyList
        emit(n, N)

p = N.pageRank / N. adjacencyList.size()
for m in N.adjacencyList {
emit(m, p)
}
reduce(id m/n, [p1,p2,…]){
s = 0
M = NULL
for p in [p1,p2,…] {
        if isVertex(p){
                M = p
        }
        else {
        s+=p
        }
    M.pageRank = (1 – alpha) / size of V + alpha * s
    emit(m, M)
    }
```

Iterate above MapReduce for number of iterations

I solved the dangling pages problem above from using a dummy vertex or “sink”. It collects contributions from all dangling nodes and redistributes the pr to all non-dangling nodes.