**Name**: Rushabh Shah
**Homework**: 4
**Github Repository**: https://github.ccs.neu.edu/rushabh0812/HW4

**Describe briefly how each step of your program is transforming the data. Be precise: show the schema of each RDD or DataSet (i.e., the "table" header) and briefly state what type of information each record (i.e., "row") stores**

1. Read the file using sc.textFile() function
2. Pass the data that is read from file to parser (written in Java) and then use filter to eliminate nulls
3. The output from the above step is split based on certain delimiter("######" in our case) to get List[Strings]. The output of this step is (String,List[String]) indicating pair (pageName, adjacencyList).
4. Adding the missing dangling nodes and combine the results.
5. Flatten the RDD by converting the RDD to pair RDD
6. Run the reduce job for generating (page, adjacencyList) and saving it in memory for future iterations.
7. Creating a default page Rank and then join it to the (page, adjacencyList)
8. Iterate over 10 iterations to calculate page rank for each node
9. Once the page ranks for all nodes is obtained ,we calculate topK nodes that have highest page rank . This is done by **.top(100)** function

**For each step, state if the dependency is narrow (no shuffling) or wide (shuffling). How many stages does your Spark have?**

**Narrow:**

```
val data = sc.textFile(args(0)+"/*.bz2").
    map(line => WikiParser.makeGraph(line)).
    filter(graphData => graphData != null).
    map(nodeName => nodeName.split("######")).
    map(name2 => (name2(0),
    if (name2.size > 1) name2(1).split(", ").toList else List[String]())).
    map(node => List((node._1, node._2)) ++ node._2.
     map(adjNode => (adjNode, List[String]()))). // Create missing dangling nodes
```

**Wide**:

```
flatMap(node => node).
    reduceByKey((x, y) => (x ++ y)).
    persist()
```

**Narrow**:

```
val ranks = data.map(node => (node._1, 1.0 / pageCount))

val delta = graph.filter(node => node._2._1.length == 0).
     reduce((a, b) => (a._1, (a._2._1, a._2._2 + b._2._2)))._2._2
```

**Wide**:

```
val page_ranks = graph.values
      .map(adjNodes => (adjNodes._1.map(node => (node, adjNodes._2 /
adjNodes._1.size)))).
      flatMap(node => node).
      reduceByKey((x, y) => x + y)
```

**Narrow**:

```
graph = data.leftOuterJoin(page_ranks).
      map(n1 => {
        (n1._1, (n1._2._1, n1._2._2 match {
          case None => (alpha / pageCount) + ((alpha_rem) * delta / pageCount)
          case Some(x: Double) => (alpha / pageCount) + (alpha_rem * ((delta / pageCount) +
x))
        }))
      })
```

**Performance Comparison**

| | **Hadoop** | **Spark** |
|---|---|---|
| **6 machines** | **2802** | **3340** |
| **11 machines** | **1750** | **2860** |

**Discuss which system is faster and briefly explain what could be the main reason for this performance difference?**

**Ans:** As seen from the above comparison, Hadoop is faster.

One of the reason could be, due to insufficient proficiency in Scala language I am not able to completely understand and persist RDD the way they are meant to be. Partitioning, reduce and map methods maybe used inefficiently. This might be slowing the process.

Additionally, a large part of the data load and processing is done in the Wiki Parses java file. If, Spark infrastructure does not by default run Java program parallel, this would lead to a bottleneck. At this stage I can say, I tired my best to implement code as efficiently as possible with my current abilities. There is scope for improvement.

**Top 100 Page Ranks:**

(0.0028956123400972,United_States_09d4)
(0.0025837662658429227,2006)
(0.0013756641642122787,United_Kingdom_5ad7)
(0.0011887559248941 93,2005)
(9.331545921987518E-4,Biography)
(9.00143865945218E-4,Canada)
(8.946269909634948E-4,England)

(8.819707744050026E-4,France)
(8.274616141252904E-4,2004)
(7.562886831203112E-4,Germany)
(7.364585608865749E-4,Australia)
(7.160204348160969E-4,Geographic_coordinate_system)
(6.664083680801913E-4,2003)
(6.491299090540319E-4,India)
(6.337472702455914E-4,Japan)
(5.378696331119127E-4,Italy)
(5.361487027053223E-4,2001)
(5.288554115340108E-4,2002)
(5.261088552781753E-4,Internet_Movie_Database_7ea7)
(5.05338627655328E-4,Europe)
(5.015343994400534E-4,2000)
(4.8157318146452093E-4,World_War_II_d045)
(4.6775567363598915E-4,London)
(4.479541246273305E-4,Population_density)
(4.4584862738815027E-4,Record_label)
(4.4299247468203176E-4,1999)
(4.399097826117927E-4,Spain)
(4.3956933153866664E-4,English_language)
(4.146181651556693E-4,Race_(United_States_Census)_a07d)
(4.136495368823275E-4,Russia)
(4.0666077969001214E-4,Wiktionary)
(3.8530050980075336E-4,Wikimedia_Commons_7b57)
(3.8278123610763475E-4,1998)
(3.751212766256924E-4,Music_genre)
(3.6565269841030886E-4,1997)
(3.610261602066143E-4,Scotland)
(3.6045735876055534E-4,New_York_City_1428)
(3.438433136340414E-4,Football_(soccer))
(3.4312474248817553E-4,1996)
(3.3844560924704663E-4,Television)
(3.3800039351019603E-4,Sweden)
(3.2696341156613685E-4,Square_mile)
(3.262717267436495E-4,Census)
(3.2299669185887405E-4,1995)
(3.216596795047187E-4,California)
(3.164946807590626E-4,China)
(3.113372532370302E-4,Netherlands)
(3.107236747236988E-4,New_Zealand_2311)
(3.0848837109117454E-4,1994)
(2.936718608554162E-4,1991)
(2.9131985809121593E-4,1993)
(2.89386728132904E-4,1990)
(2.8901262991085194E-4,New_York_3da4)
(2.8844123039069357E-4,Public_domain)
(2.790616339903877E-4,1992)
(2.787788273348963E-4,United_States_Census_Bureau_2c85)

(2.7782083155419586E-4,Film)
(2.759587905172543E-4,Scientific_classification)
(2.7540669173526934E-4,Actor)
(2.725732034027786E-4,Norway)
(2.71679957615189E-4,Ireland)
(2.649319762745635E-4,Population)
(2.617771298000105E-4,1989)
(2.557897929509219E-4,1980)
(2.5559028895676215E-4,Marriage)
(2.5503294304771233E-4,January_1)
(2.5429178563588336E-4,Brazil)
(2.529464259863485E-4,Mexico)
(2.5191426125792903E-4,Latin)
(2.490248595071528E-4,1986)
(2.469965868947967E-4,Politician)
(2.427784162208756E-4,1985)
(2.4240004633216882E-4,1979)
(2.4189489896314504E-4,1982)
(2.4173203771582006E-4,French_language)
(2.41570354257894E-4,1981)
(2.411754522508159E-4,Per_capita_income)
(2.3980906233639753E-4,Poland)
(2.394875228472537E-4,1974)
(2.3937069933951236E-4,Album)
(2.377985660983332E-4,South_Africa_1287)
(2.3722845366140895E-4,Switzerland)
(2.371483689076466E-4,1984)
(2.3714758540815131E-4,1987)
(2.3698196286984653E-4,1983)
(2.3576252382941844E-4,Record_producer)
(2.331483660262823E-4,1970)
(2.3176630161821717E-4,1988)
(2.304074207449563E-4,1976)
(2.2916979771211816E-4,$Km^2$)
(2.2749905467795402E-4,1975)
(2.247856814854485E-4,1969)
(2.245462310587684E-4,Paris)
(2.2348383281851965E-4,Greece)
(2.232404651902856E-4,1972)
(2.224573575154661E-4,Personal_name)
(2.2197817108528283E-4,1945)
(2.2135127507557308E-4,1977)
(2.2040350997556632E-4,Poverty_line)
(2.2034742559349092E-4,1978)