Foster Mosden
Assignment 7 - Big Data

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
sc = spark.sparkContext
lines = sc.textFile("s3://fm-ga2/links.txt")
links = lines.map(lambda url_line: url_line.split(" ")).groupByKey()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

def computeContribs(urls, rank):
      num_urls = len(urls)
      for url in urls:
                  yield (url, rank / num_urls)

for iteration in range(1):
      contribs = links.join(ranks).flatMap(lambda url_urls_rank:
                  computeContribs(url_urls_rank[1][0],
url_urls_rank[1][1]))
      ranks = contribs.reduceByKey(lambda x,y:x+y).mapValues(lambda rank:
rank * 0.85 + 0.15)

print(ranks.collect())
```

1 Iteration:

Dampening/Page	**.15**	**.50**	**.85**
A	1.125	1.416	1.708
B	.975	0.916	0.8583
C	1.05	1.16	1.283

85% Contribution:

Iterations/Page	**1**	**5**	**10**	**20**
A	1.708	1.4965	1.3857	1.3154
B	0.8583	0.7993	0.7455	0.7103
C	1.283	1.1479	1.0656	1.0129

I should note that I did this in a rush while getting on/actually onboard a flight and I am actually pretty sure I might have inverted some of these values in my haste.

From my findings, it appears that adding a damping factor increased the relative rank of those pages that had a lot of relationships pointing into them. On the other hand, those pages with a lot of relationships pointing outward, but few pointing in, had their relative rank drop.

Iterating over each page more seemed to decrease the rank for all of our letters. However, this iteration is important to getting to the “true” pagerank of each letter, as it’s an iterative algorithm. In reality, my estimations using 1 iteration and varied damping factors is very much an estimate and likely quite far from the “true” pagerank. The more I iterated, the closer I got to the true value.

I suppose my findings are that only iterating once won’t get you a very accurate rank, but isn’t terribly far off when it comes to deducing a relative rank within a small subset of pages (or in this case, letters). However, I assume that these small differences as you continue to iterate make a more profound difference when you’re deaiing with a greater number of webpages (ie every webpage in existence).

My other finding is that the damping factor makes a profound difference when it comes to rank. Again, this didn’t have a big impact with a small subset of pages, but surely would with a greater number. The damping is important because it points out the weakness of through links as you go through and through.

1 iteration at .85

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
sc = spark.sparkContext
lines = sc.textFile("s3://fm-ga2/links.txt")
links = lines.map(lambda url_line: url_line.split(" ")).groupByKey()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

def computeContribs(urls, rank):
        num_urls = len(urls)
        for url in urls:
                yield (url, rank / num_urls)

for iteration in range(1):
        contribs = links.join(ranks).flatMap(lambda url_urls_rank:
                computeContribs(url_urls_rank[1][0], url_urls_rank[1][1]))
        ranks = contribs.reduceByKey(lambda x,y:x+y).mapValues(lambda rank: rank * 0.85 + 0.15)

print(ranks.collect())
```

```
[('B', 0.8583333333333333), ('A', 1.708333333333333), ('C', 1.2833333333333332)]
```

1 iteration at .50

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
sc = spark.sparkContext
lines = sc.textFile("s3://fm-ga2/links.txt")
links = lines.map(lambda url_line: url_line.split(" ")).groupByKey()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

def computeContribs(urls, rank):
        num_urls = len(urls)
        for url in urls:
                yield (url, rank / num_urls)

for iteration in range(1):
        contribs = links.join(ranks).flatMap(lambda url_urls_rank:
                computeContribs(url_urls_rank[1][0], url_urls_rank[1][1]))
        ranks = contribs.reduceByKey(lambda x,y:x+y).mapValues(lambda rank: rank * 0.5 + 0.5)

print(ranks.collect())
```

```
[('B', 0.9166666666666666), ('A', 1.4166666666666665), ('C', 1.1666666666666665)]
```

1 iteration at .15

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
sc = spark.sparkContext
lines = sc.textFile("s3://fm-ga2/links.txt")
links = lines.map(lambda url_line: url_line.split(" ")).groupByKey()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

def computeContribs(urls, rank):
        num_urls = len(urls)
        for url in urls:
                yield (url, rank / num_urls)

for iteration in range(1):
        contribs = links.join(ranks).flatMap(lambda url_urls_rank:
                computeContribs(url_urls_rank[1][0], url_urls_rank[1][1]))
        ranks = contribs.reduceByKey(lambda x,y:x+y).mapValues(lambda rank: rank * 0.15 + 0.85)

print(ranks.collect())
```

```
[('B', 0.975), ('A', 1.125), ('C', 1.05)]
```

5 iterations at 85% contribution

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
sc = spark.sparkContext
lines = sc.textFile("s3://fm-ga2/links.txt")
links = lines.map(lambda url_line: url_line.split(" ")).groupByKey()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

def computeContribs(urls, rank):
        num_urls = len(urls)
        for url in urls:
                yield (url, rank / num_urls)

for iteration in range(5):
        contribs = links.join(ranks).flatMap(lambda url_urls_rank:
                computeContribs(url_urls_rank[1][0], url_urls_rank[1][1]))
        ranks = contribs.reduceByKey(lambda x,y:x+y).mapValues(lambda rank: rank * 0.85 + 0.15)

print(ranks.collect())
```

```
[('A', 1.4965020996093743), ('C', 1.147901770833333), ('B', 0.7993014420572915)]
```

10 iterations at 85%

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
sc = spark.sparkContext
lines = sc.textFile("s3://fm-ga2/links.txt")
links = lines.map(lambda url_line: url_line.split(" ")).groupByKey()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

def computeContribs(urls, rank):
        num_urls = len(urls)
        for url in urls:
                yield (url, rank / num_urls)

for iteration in range(10):
        contribs = links.join(ranks).flatMap(lambda url_urls_rank:
                computeContribs(url_urls_rank[1][0], url_urls_rank[1][1]))
        ranks = contribs.reduceByKey(lambda x,y:x+y).mapValues(lambda rank: rank * 0.85 + 0.15)

print(ranks.collect())
```

```
[('B', 0.7455188698656946), ('C', 1.0656248014469072), ('A', 1.3857307330281197)]
```

20 iterations at 85%

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
sc = spark.sparkContext
lines = sc.textFile("s3://fm-ga2/links.txt")
links = lines.map(lambda url_line: url_line.split(" ")).groupByKey()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

def computeContribs(urls, rank):
        num_urls = len(urls)
        for url in urls:
                yield (url, rank / num_urls)

for iteration in range(20):
        contribs = links.join(ranks).flatMap(lambda url_urls_rank:
                computeContribs(url_urls_rank[1][0], url_urls_rank[1][1]))
        ranks = contribs.reduceByKey(lambda x,y:x+y).mapValues(lambda rank: rank * 0.85 + 0.15)

print(ranks.collect())
```

```
[('C', 1.0129198436948377), ('A', 1.3154720694847448), ('B', 0.7103676179049304)]
```