https://towardsdatascience.com/graphs-and-paths-pagerank-54f180a1aa0a
(https://towardsdatascience.com/graphs-and-paths-pagerank-54f180a1aa0a)

In [ ]:

```
import findspark
findspark.init()
import pyspark
```

In [ ]:

```
sc = pyspark.SparkContext(appName="PageRank")
```

In [ ]:

```
def computeContribe(urls, rank):
    """calculates URL contributions to the rank of others URLs"""
    num_urls = len(urls)
    contribs = []
    for url in urls:
        contribs.append((url, rank/num_urls))
    return contribs
```

In [ ]:

```
graph = sc.textFile("C:/Users/EASYFRONT/Desktop/bigdata/seance02/pagerank/web-Google.tx
t")
#clean up graph rdd from comment and parse it
graph_cup = graph.filter(lambda x : "#" not in x).map(lambda x : x.split("\t"))
graph_cup.collect()
```

In [ ]:

```
graph_cup.getNumPartitions()
```

In [ ]:

```
#draft algorithm
links = graph_cup.groupByKey().partitionBy(100).cache()
#initialize the ranks to 1 for each URL
ranks = links.map(lambda url_neighbors : (url_neighbors[0],1.0) )
#ranks.collect()
for iteration in range(2):
    contribs = links.join(ranks).flatMap(lambda url_urls_rank : computeContribe(url_url
s_rank[1][0],url_urls_rank[1][0]))
    ranks = contribs.reduceByKey(lambda x,y : x+y).mapValues(lambda rank : rank*0.85 +
0.15)

for (link,rank) in ranks.sortBy(lambda x:-x[1]).take(10):
    print("%s has rank : %s." % (link, rank))
```

In [ ]:

```
sc.stop()
```

SMALL GRAPH EXAMPLE

In [ ]:

```
rdd_medium_example = sc.parallelize([("Amzn","Twtr"),("Amzn","Medm"),("Fb","Amzn"),("F
b","Twtr"),("Medm","Twtr"),("Medm","Amzn"),("Medm","Mspc"),("Medm","Fb"),("Twtr","Medm"
),("Mspc","Twtr")])
rdd_medium_example.collect()
```

In [ ]:

```
nodes = rdd_medium_example.flatMap(lambda x : [x[0]]+[x[1]]).distinct()
nbnodes = nodes.count()
```

In [ ]:

```
#initialization phase
ranks = nodes.map(lambda x : (x,1/nbnodes))
ranks.collect()
```

In [ ]:

```
#compute following structure (node, all vertices that comme out of from node (aretes so
rtantes))
links = rdd_medium_example.groupByKey()
for iteration in range(2):
    contribs = links.join(ranks).flatMap(lambda url_urls_rank : computeContribe(url_url
s_rank[1][0],url_urls_rank[1][1]))
    ranks = contribs.reduceByKey(lambda x,y : x+y).mapValues(lambda rank : round(rank*
0.85 + 0.15/nbnodes,2))

for (link,rank) in ranks.sortBy(lambda x:-x[1]).collect():
    print("%s has rank : %s." % (link, rank))
```

DEMISTIFYING ONE ITERATION

In [ ]:

```
links = rdd_medium_example.groupByKey()
links = links.map(lambda x : (x[0], list(x[1])) )
links.collect()
```

In [ ]:

```
links_ranks = links.join(ranks)
links_ranks.collect()
```

In [ ]:

```
contribs = links_ranks.flatMap(lambda url_urls_rank : computeContribe(url_urls_rank[1][
0],url_urls_rank[1][1]))
contribs.collect()
```

In [ ]:

```
ranks_it1 = contribs.reduceByKey(lambda x,y : x+y).mapValues(lambda rank : rank*0.85 +
0.15/nbnodes)
ranks_it1.collect()
```

In [ ]:

```
links2 = rdd_medium_example.groupByKey()
links2 = links2.map(lambda x : (x[0], list(x[1])) )
links2 = links2.mapValues(lambda x : 1)
links2.collect()
```