I assign initial rank 0 as: [A: 1.7, B:0.9, C:1.3, D:0.2]

Table

Description automatically generatedTables:

Table

Description automatically generated

FINDINGS:

1. Node d is neglected because it has no in-coming links, whicn means it’s pagerank is 0
2. The higher the damping factor, the closer the pagerank is to the original value of each node (1.7, 0.9, 1.3) in 1 iteration. This is because the initial ranks of A,B,C and their links form a relatively stable system, that is, the new ranks based on the in-coming ranks is close to the old rank. Also, the higher the controbution, the larger the sum of the ranks. This is because the original ranks sum up larger than 4, so the incoming ranks sum up larger than four, also node D is out of the system, so the pagerank algorithm actually want to the sum of A,B,C’s ranks to 3, so it’s decreasing the sum in each iteration. Therefore, the higher the contribution, the less the decrease, the larger the sum.
3. For 0.85 contribution, the more the iterations, the lower the value of each node A,B,C. There is a “energy loss” during each iteration. This is because the sum of initial rank = 0.9+0.2+1.3+1.7 = 4.1 > 4, so every time it uses the formula : rank \* 0.85 + 0.15, the average rank will be : the old average rank\*0.85 + 0.15. since the average rank is always no less than 1, the avearage rank will decrease until it converges to 1.

CODE:

lines = sc.textFile("s3://671happyzhua273/pagerank.txt")

links = lines.map(lambda url\_line: url\_line.split(" ")).groupByKey()

ranks = sc.parallelize([("A", 1.7), ("B", 0.9), ("C", 1.3), ("D", 0.2)])

def computeContribs(urls, rank):

num\_urls = len(urls)

for url in urls:

yield (url, rank / num\_urls)

def pagerank(itreration\_num,contribution,links,ranks):

for iteration in range(itreration\_num):

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 \* contribution + 1 - contribution)

df = spark.createDataFrame(data=ranks,

schema = ["node\_name",str(itreration\_num)+" iteration + "+str(contribution)+" damp"])

return(df)

a = pagerank(1,0.15,links,ranks)

b = pagerank(1,0.5,links,ranks)

c = pagerank(1,0.85,links,ranks)

ab = a.join(b, ['node\_name'])

abc = ab.join(c,['node\_name'])

abc.show()

e = pagerank(1,0.85,links,ranks)

f = pagerank(5,0.85,links,ranks)

g = pagerank(10,0.85,links,ranks)

h = pagerank(20,0.85,links,ranks)

ef = e.join(f, ['node\_name'])

efg = ef.join(g,['node\_name'])

efgh = efg.join(h,['node\_name'])

efgh.show()