1. Introduction

Pagerank is an algorithm used by google search to rank web pages in their search results. It estimates how important a result is by counting the number and quality of links to a page.

1.1 Dangling Nodes (δ)

Some nodes do not point to any other node (basically a dead end), these nodes are called dangling nodes.

1.2 Convergence

This is the point when the ranks of all nodes do not experience much change when compared to the previous ranking. The final ranks are decided when convergence is reached. A convergence threshold is set in case a state where 0 change cannot be achieved.

1.3 Damping Factor (α)

It is the probability that a person will continue following the links is damping factor. The probability that they jump to another page is (1 - α).

2. Methodologies

2.1 Pagerank Equation

$$P(n) = \alpha \frac{1}{|V|} + (1 - \alpha)\left(\frac{\delta}{|V|} + \sum_{m \in L(n)} \frac{P(m)}{C(m)}\right)$$

P(n) -> Pagerank of current iteration

α -> Damping factor

δ -> Sum of ranks of dangling node

|V| -> number of nodes

P(m) -> previous rank

C(m) -> number of neighbour nodes

2.2. Dangling Nodes

The dangling nodes are first identified using the “deltaCheck()” function and all the rank values of the dangling nodes are added and stored in the “delta” variable. This variable is later used in the pagerank equation.

```
# to check if node is a dangling node
def deltaCheck(element):
  if (len(element[1][1])==0):
    return element[1][0]
  return 0
```

```
# Delta is the sum of the ranks of dangling nodes
delta = joinedRDD.map(lambda element: deltaCheck(element)).reduce(add)
print("delta:",delta)
```

```
# The pagerank formula
def prFormula(element):
  global delta
  pageRankValue = (alpha/noOfNodes) + (1 - alpha)*(delta/noOfNodes + element[1])
  return ((element[0], pageRankValue))
```

2.3 Multiple Iterations

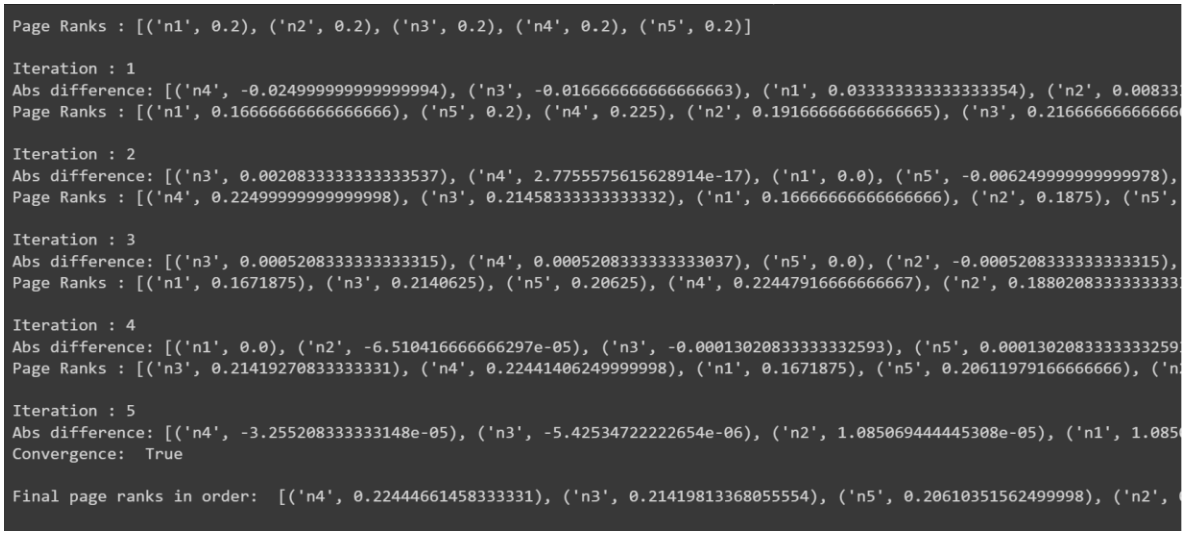
The code can have as many iterations as it needs. The function “convergenceCheck()” compares the previous ranks and present ranks to decide when optimum ranks or convergence has been achieved. Once convergence is achieved, the code exits the loop.

```
# To check convergence between current and previous ranks
def convergenceCheck( prevRDD, newRDD):
  global threshold
  absList = (prevRDD.union(newRDD)).map(lambda element: element).reduceByKey(sub)
  abovethreshold = absList.filter(lambda element: abs(element[1]) > threshold )
  if (abovethreshold.count() < 1):
    return absList, 1
  return absList, 0
```

```
# Absolute value of difference between present and previous rank to calculate convergence
absList, convergence = convergenceCheck(previousRanks, pageRankRDD)
print("Abs difference:",absList.collect())
# Break iteration loop if convergence is achieved
# delta=0
if (convergence == 1):
  break
```

3. Snapshots of Sample runs:-

3.1 Pagerank for 5 nodes

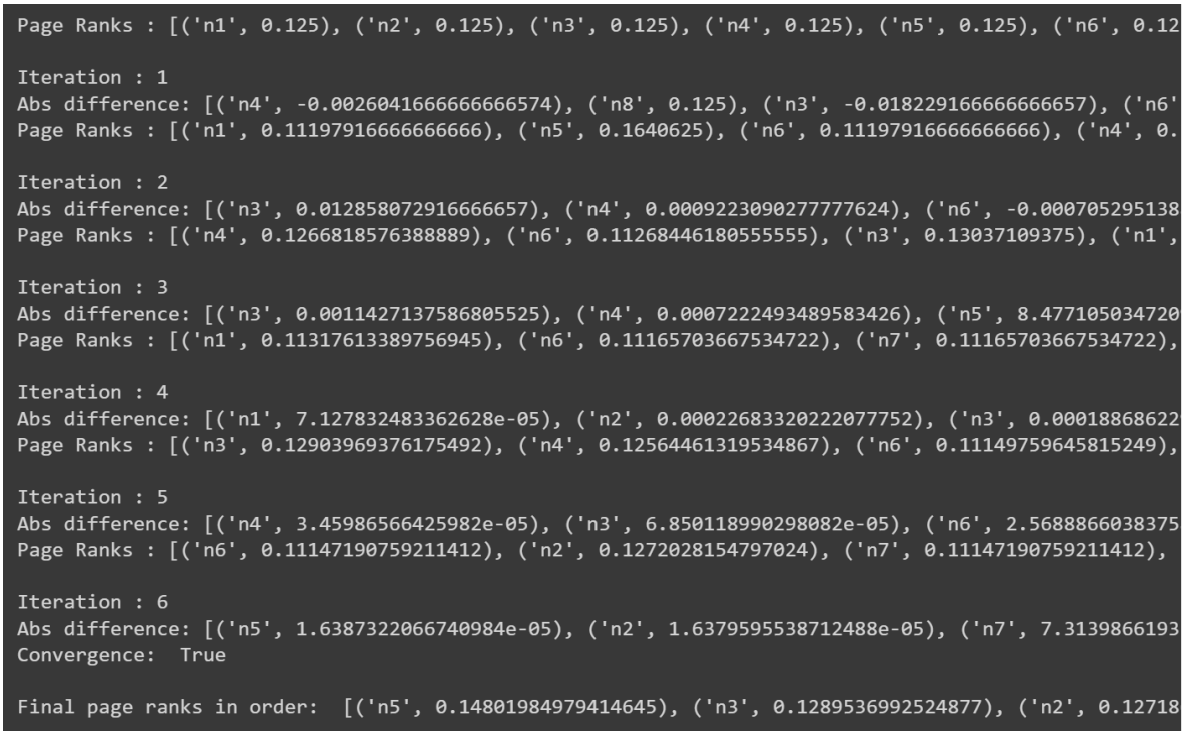


```
Page Ranks : [('n1', 0.2), ('n2', 0.2), ('n3', 0.2), ('n4', 0.2), ('n5', 0.2)]

Iteration : 1
Abs difference: [('n4', -0.024999999999999994), ('n3', -0.016666666666666663), ('n1', 0.033333333333333354), ('n2', 0.00833
Page Ranks : [('n1', 0.16666666666666666), ('n5', 0.2), ('n4', 0.225), ('n2', 0.19166666666666665), ('n3', 0.216666666666666

Iteration : 2
Abs difference: [('n3', 0.0020833333333333537), ('n4', 2.7755575615628914e-17), ('n1', 0.0), ('n5', -0.006249999999999978),
Page Ranks : [('n4', 0.22499999999999998), ('n3', 0.21458333333333332), ('n1', 0.16666666666666666), ('n2', 0.1875), ('n5',

Iteration : 3
Abs difference: [('n3', 0.0005208333333333315), ('n4', 0.0005208333333333037), ('n5', 0.0), ('n2', -0.0005208333333333315),
Page Ranks : [('n1', 0.1671875), ('n3', 0.2140625), ('n5', 0.20625), ('n4', 0.22447916666666667), ('n2', 0.18802083333333331

Iteration : 4
Abs difference: [('n1', 0.0), ('n2', -6.510416666666297e-05), ('n3', -0.0001302083333333259
Page Ranks : [('n3', 0.21419270833333331), ('n4', 0.22441406249999998), ('n1', 0.1671875), ('n5', 0.20611979166666666), ('n

Iteration : 5
Abs difference: [('n4', -3.255208333333148e-05), ('n3', -5.42534722222654e-06), ('n2', 1.085069444445308e-05), ('n1', 1.085
Convergence:  True

Final page ranks in order:  [('n4', 0.22444661458333331), ('n3', 0.21419813368055554), ('n5', 0.20610351562499998), ('n2',
```

Time taken = 49s

Number of iterations = 5

3.2 Pagerank for 8 nodes with dangling nodes



```
Page Ranks : [('n1', 0.125), ('n2', 0.125), ('n3', 0.125), ('n4', 0.125), ('n5', 0.125), ('n6', 0.12

Iteration : 1
Abs difference: [('n4', -0.0026041666666666574), ('n8', 0.125), ('n3', -0.018229166666666657), ('n6'
Page Ranks : [('n1', 0.11197916666666666), ('n5', 0.1640625), ('n6', 0.11197916666666666), ('n4', 0.

Iteration : 2
Abs difference: [('n3', 0.012858072916666657), ('n4', 0.0009223090277777624), ('n6', -0.000705295138
Page Ranks : [('n4', 0.1266818576388889), ('n6', 0.11268446180555555), ('n3', 0.13037109375), ('n1',

Iteration : 3
Abs difference: [('n3', 0.0011427137586805525), ('n4', 0.0007222493489583426), ('n5', 8.477105034720
Page Ranks : [('n1', 0.11317613389756945), ('n6', 0.11165703667534722), ('n7', 0.11165703667534722),

Iteration : 4
Abs difference: [('n1', 7.127832483362628e-05), ('n2', 0.00022683320222077752), ('n3', 0.0001886862
Page Ranks : [('n3', 0.12903969376175492), ('n4', 0.12564461319534867), ('n6', 0.11149759645815249),

Iteration : 5
Abs difference: [('n4', 3.45986566425982e-05), ('n3', 6.850118990298082e-05), ('n6', 2.5688866038375
Page Ranks : [('n6', 0.11147190759211412), ('n2', 0.1272028154797024), ('n7', 0.11147190759211412),

Iteration : 6
Abs difference: [('n5', 1.6387322066740984e-05), ('n2', 1.6379595538712488e-05), ('n7', 7.3139866193
Convergence:  True

Final page ranks in order:  [('n5', 0.14801984979414645), ('n3', 0.1289536992524877), ('n2', 0.12718
```

Time taken = 72s

Number of iterations = 6

3.3 Pagerank for 20 nodes with dangling nodes

```
Page Ranks : [('n1', 0.05), ('n2', 0.05), ('n3', 0.05), ('n4', 0.05), ('n5', 0.05), ('n6', 0.05), ('n7', 0.05), (

Iteration : 1
Abs difference: [('n4', 0.0016666666666666705), ('n8', -0.0029166666666666646), ('n11', -0.0024999999999999953),
Page Ranks : [('n1', 0.045208333333333336), ('n5', 0.05499999999999999), ('n14', 0.04833333333333333), ('n17', 0.

Iteration : 2
Abs difference: [('n8', 0.0008862847222222275), ('n15', -0.0014053819444444504), ('n3', -0.0006553819444444428),
Page Ranks : [('n18', 0.05600607638888888), ('n16', 0.0424296875), ('n8', 0.05203038194444444), ('n4', 0.0487109

Iteration : 3
Abs difference: [('n3', 3.764467592592968e-05), ('n4', -9.480613425925932e-05), ('n5', 4.524016203703851e-05), ('
Page Ranks : [('n1', 0.04545170355902778), ('n16', 0.04244427083333333), ('n15', 0.06350715060763888), ('n6', 0.0

Iteration : 4
Abs difference: [('n8', 1.2157298900916658e-07), ('n11', 1.2736002603930263e-07), ('n9', -3.57272677951187e-06),
Convergence:  True

Final page ranks in order:  [('n15', 0.06351915739836517), ('n3', 0.05727475870768229), ('n18', 0.056047477710865
```

Time taken = 41s

Number of iterations = 4

3.4 Pagerank for 30 nodes with dangling nodes

```
Page Ranks : [('n1', 0.03333333333333333), ('n2', 0.03333333333333333), ('n3', 0.03333333333333333), ('n4', 0.03333

Iteration : 1
Abs difference: [('n4', 0.00013888888888888978), ('n8', -0.0036111111111111135), ('n11', -0.004027777777777783), ('
Page Ranks : [('n1', 0.029722222222222223), ('n30', 0.028333333333333335), ('n5', 0.04069444444444445), ('n14', 0.0

Iteration : 2
Abs difference: [('n8', 0.0006782407407407431), ('n15', -0.0013414351851851816), ('n3', -0.0006296296296296328), ('
Page Ranks : [('n18', 0.039467592592592596), ('n23', 0.033240740740740744), ('n11', 0.03778935185185185), ('n16', 0

Iteration : 3
Abs difference: [('n3', 6.154031635802193e-05), ('n4', -5.063175154321198e-05), ('n27', -3.0087770061727748e-05), (
Page Ranks : [('n16', 0.029681635802469138), ('n1', 0.02993587962962963), ('n15', 0.045673259066358024), ('n20', 0.

Iteration : 4
Abs difference: [('n22', 3.3247331532940128e-06), ('n8', -9.110162680001088e-07), ('n23', -3.1621736754083862e-06),
Convergence:  True

Final page ranks in order:  [('n15', 0.045677765701356746), ('n5', 0.04050441364052855), ('n18', 0.0394861048016332
```

Time taken = 37s

Number of iterations = 4

4. Observations

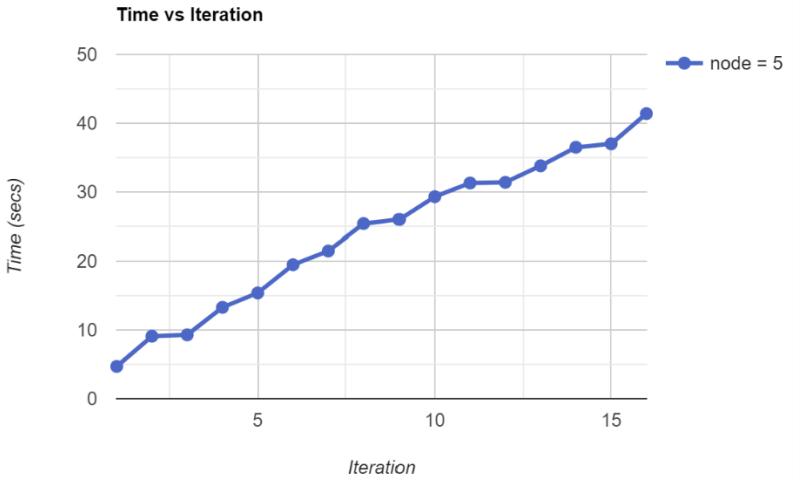
4.1 Computation Time & number of nodes.

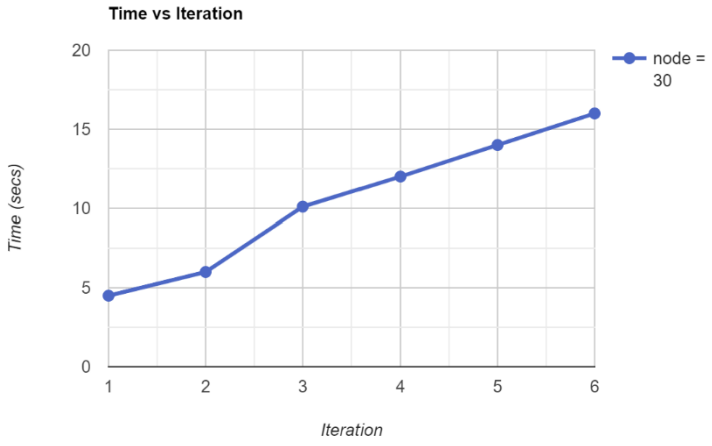
Number of Nodes	Number of iterations	Computation Time (sec)
5	5	49
8	6	72
20	4	41
30	4	37
α = 0.75		Threshold = 0.001

The above results show that the computation speed is not dependent on the size of the data being assessed.

4.2 Computation Time & Iterations.

Below computations were performed with α = 0.15 computing for 5 nodes and 30 nodes respectively





The above graphs show that the computation time increases every subsequent iteration and is independent of number of nodes and damping factor.

4.3 Damping Factor vs Iterations.

The below experiments were conducted by increasing and decreasing the damping factor α.

Number of Nodes	Number of iterations	Computation Time (sec)
5	16	373
30	6	66
α = 0.15		Threshold = 0.001

Number of Nodes	Number of iterations	Computation Time (sec)
5	9	135
30	5	50
α = 0.35		Threshold = 0.001

Number of Nodes	Number of iterations	Computation Time (sec)
5	5	49
30	4	37
α = 0.75		Threshold = 0.001

The number of iterations decrease as the damping factor is decreased. Which also increases the computation time.

Increasing damping factor -> increases the probability of user following the links

5. Conclusion

1. Increasing the number of nodes, or the size of data is inconsequential when it comes to overall computation time.
2. The computation time of iterations increases for newer iterations.
3. The rate of convergence increases for ideal damping factor, which means fewer iterations and faster computation. Finding the ideal damping factor is key.

Damping factor = 0.85 is considered the ideal/default damping factor