**Aim: To implement Page Rank Algorithm**

**Objective:-Develop a program to implement page rank algorithm**

**Theory:-**PageRank (PR) is an algorithm used by Google Search to rank web pages in their search engine results. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. Page Rank Algorithm is designed to increase the effectiveness of search engines and improve their efficiency. It is a way of measuring the importance of website pages. Page rank is used to prioritize the pages returned from a traditional search engine using keyword searching. Page rank is calculated based on the number of pages that point to it. The value of the page rank is the probability will be between 0 and 1. A web page is a directed graph having two important components: nodes and connections. The pages are nodes and hyperlinks are the connections, the connection between two nodes.

Page rank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important website are likely to receive more links from other websites. The page rank value of individual node in a graph depends on the page rank value of all the nodes which connect to it and those nodes are cyclically connected to the nodes whose ranking we want; we use converging iterative method for assigning values to page rank. In short page rank is a vote, by all the other pages on the web, about how important a page is. A link to a page count as a vote of support. If there is no link, there is no support.

We assume that page A has pages B…….N which point to it.

Page rank of a page A is given as follows:

PR(A)=(1-β) +β ( (PR(B)/cout(B) )+ (PR(C )/cout(C ) )+-----+ (PR(N)/cout(N) ) )

Parameter β is a teleportation factor which can be set between 0 and 1. Cout(A) is defined as the number of links going out of page A.

**Code:**

import numpy as np

import math

# normalize the matrix (make it a probability matrix (all cols sum to 1))

def normalizeAdjacencyMatrix(A):

n = len(A) # n = num of rows/cols in A

for j in range(len(A[0])):

sumOfCol = 0

for i in range(len(A)):

sumOfCol += A[i][j]

if sumOfCol == 0: # adjust for dangling nodes (columns of zeros)

for val in range(n):

A[val][j] = 1/n

else:

for val in range(n):

A[val][j] = (A[val][j] / sumOfCol)

return A

# implement damping matrix using formula

# M = dA + (1-d)(1/n)Q, where Q is an array of 1's and d is the damping factor

def dampingMatrix(A):

n = len(A) # n = num of rows/cols in A

dampingFactor = 0.85

Q = [[1/n]\*n]\*n

arrA = np.array(A)

arrQ = np.array(Q)

arrM = np.add((dampingFactor)\*arrA, (1-dampingFactor)\*arrQ) # create damping matrix

return arrM

# find eigenvector corresponding to eigenvalue 1

def findSteadyState(M, n):

# find eigenvectors

evectors = np.linalg.eig(M)[1]

# find eigenvalues

eigenValues = np.linalg.eig(M)[0]

lstEVals = []

for val in eigenValues:

lstEVals.append(round(val))

# find eigenvector with eigenvalue 1

idxWithEval1 = lstEVals.index(1)

steadyStateVector = evectors[:, idxWithEval1]

# normalize steady state vector so its components sum to 1

lstVersionSteadyState = []

sumOfComps = 0

returnVector = []

for val in steadyStateVector:

sumOfComps += val

lstVersionSteadyState.append(val)

for val in lstVersionSteadyState:

returnVector.append(val/sumOfComps)

return returnVector

def pageRank(A):

n = len(A) # n = num of rows/cols in A

A = normalizeAdjacencyMatrix(A)

M = dampingMatrix(A)

# find steady state vector

steadyStateVectorOfA = findSteadyState(M, n)

return steadyStateVectorOfA

# TEST CASES

print("\nPage Rank Examples")

# 1) (corresponds to directed graph (1) on readme.md)

matrix1 = [ [0, 1, 0, 0],

[0, 0, 0, 0],

[0, 1, 0, 1],

[0, 0, 1, 0] ]

print("1) matrix 1 = ", matrix1)

print("steady state vector: ")

print(pageRank(matrix1))

# expected output: [0.077, 0.054, 0.441, 0.429]

# 2)

matrix2 = [ [0, 0, 1, 0, 0, 0, 0, 0],

[1, 0, 0, 1, 0, 0, 0, 0],

[1, 0, 0, 0, 0, 0, 0, 0],

[1, 1, 1, 0, 0, 0, 0, 0],

[0, 1, 0, 0, 0, 1, 0, 0],

[0, 0, 0, 0, 0, 0, 1, 1],

[0, 0, 0, 1, 1, 0, 0, 1],

[0, 0, 0, 0, 0, 1, 0, 0] ]

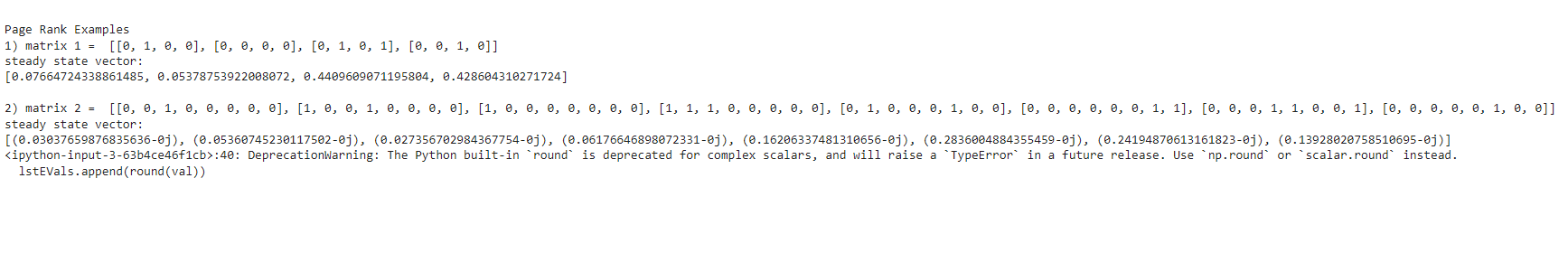
print("\n2) matrix 2 = ", matrix2)

print("steady state vector: ")

print(pageRank(matrix2))

# expected output: [0.03037, 0.0536, 0.02735, 0.0617, 0.1621, 0.2836, 0.2419, .1393]

**Output:**



**Conclusion**: The output of the program gives the steady-state vector representing the PageRank of each node, reflecting the relative importance of each webpage in the network. The PageRank algorithm is crucial for improving the ranking of search engine results by considering both the quantity and quality of incoming links, providing more relevant and efficient search outcomes.