**EXPERIMENT NO. 10**

Aim: Implementation of Page rank/HITS algorithm

Requirements: Windows/MAC/Linux O.S, Compatible version of Python, Python libraries: Numpy, Matplotlib and Networkx.

Theory:

Page Rank:

The PageRank algorithm measures the importance of each node within the graph, based on the number incoming relationships and the importance of the corresponding source nodes. The underlying assumption roughly speaking is that a page is only as important as the pages that link to it.

PageRank is introduced in the original Google paper as a function that solves the following equation:

where,

* we assume that a page *A* has pages *T1* to *Tn* which point to it.
* *d* is a damping factor which can be set between 0 (inclusive) and 1 (exclusive). It is usually set to 0.85.
* *C(A)* is defined as the number of links going out of page *A*.

This equation is used to iteratively update a candidate solution and arrive at an approximate solution to the same equation.

Considerations:

There are some things to be aware of when using the PageRank algorithm:

* If there are no relationships from within a group of pages to outside the group, then the group is considered a spider trap.
* Rank sink can occur when a network of pages is forming an infinite cycle.
* Dead-ends occur when pages have no outgoing relationship.

Changing the damping factor can help with all the considerations above. It can be interpreted as a probability of a web surfer to sometimes jump to a random page and therefore not getting stuck in sinks.

Code:

import numpy as np

import operator

import networkx as nx

import matplotlib.pyplot as plt

import pylab

trusted\_pages\_ratio = 0.4

trusted\_pages = []

maxer = 0

nodes\_dict = {}

nodes = []

count = 0

beta = 0.85

# Reading form file and then saving the data to a dictionary

# of the form {NODE: [Set of nodes being pointed to by the "NODE"]}

with open("data.txt", "r") as data\_file:

for line in data\_file:

line\_values = line.split("\t")

a = int(line\_values[0])

b = int(line\_values[1])

if a > maxer:

maxer = a

if b > maxer:

maxer = b

if a not in nodes:

nodes.append(a)

if b not in nodes:

nodes.append(b)

if a not in nodes\_dict:

nodes\_dict[a] = [b]

else:

nodes\_dict[a].append(b)

# M is the Transition Matrix

# v is the matrix that defines the probability of the random surfer of being at any paricular node

M = np.zeros((maxer+1, maxer+1))

v = np.zeros(maxer + 1)

# Defining the Transition matrix

for from\_node in nodes\_dict:

length = len(nodes\_dict[from\_node])

fraction = 1/length

for to\_node in nodes\_dict[from\_node]:

M[to\_node][from\_node] = fraction

# Defining initial v matrix

no\_of\_nodes = len(nodes)

fraction = 1 / no\_of\_nodes

for i in range(1, maxer + 1):

if i in nodes:

v[i] = fraction

# Definining the teleport matrix which takes care of the Dead ends and Spider traps

teleport = (1 - beta) \* v

M = beta \* M

# Carrying out the iterations until matrix v stops changing

while(1):

v1 = np.dot(M, v) + teleport

if np.array\_equal(v1,v):

break

else:

v = v1

count += 1

print("No. of iterations required without considering TrustRank: " + str(count))

# Sorting nodes with respect to the final ranks of the nodes

page\_rank\_score = []

for i in range(1, len(v)):

if v[i] != 0:

page\_rank\_score.append([i, v[i]])

sorted\_page\_rank\_score = sorted(page\_rank\_score, key = operator.itemgetter(1), reverse = True)

# Incorporating the concept of Trust rank

no\_of\_trusted\_pages = int(trusted\_pages\_ratio \* len(sorted\_page\_rank\_score))

trusted\_pages = [page\_info[0] for i, page\_info in zip(range(0, no\_of\_trusted\_pages), sorted\_page\_rank\_score)]

fraction = 1 / no\_of\_trusted\_pages

# Defining initial v matrix

v = np.zeros(maxer + 1)

for i in range(1, maxer + 1):

if i in trusted\_pages:

v[i] = fraction

# Defining teleport set

teleport = (1 - beta) \* v

count = 0

# Carrying out the iterations until matrix v stops changing

while(1):

v1 = np.dot(M, v) + teleport

if np.array\_equal(v1,v):

break

else:

v = v1

count += 1

print("No. of iterations required after considering TrustRank: " + str(count))

# Sorting nodes with respect to the final ranks of the nodes

page\_rank\_score\_after\_trustrank = []

for i in range(1, len(v)):

if v[i] != 0:

page\_rank\_score\_after\_trustrank.append([i, v[i]])

sorted\_page\_rank\_score\_after\_trustrank = sorted(page\_rank\_score\_after\_trustrank, key = operator.itemgetter(1), reverse = True)

# Plotting the top 30 nodes

nodes\_for\_graph = []

edge\_list\_for\_graph = []

G = nx.DiGraph()

for i, page\_info in zip(range(30), sorted\_page\_rank\_score\_after\_trustrank):

nodes\_for\_graph.append(page\_info[0])

# print(page\_info[1])

print(nodes\_for\_graph)

for node in nodes\_for\_graph:

if node in nodes\_dict:

for page in nodes\_dict[node]:

if page in nodes\_for\_graph:

edge\_list\_for\_graph.append([node, page])

G.add\_edges\_from(edge\_list\_for\_graph)

edge\_colors = ['grey' for edge in G.edges]

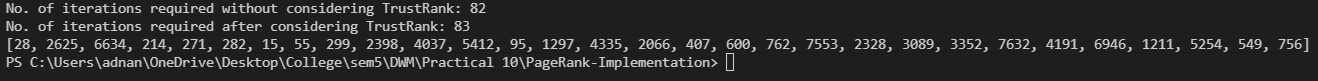
final\_node\_size = [1950000 \* page\_info[1] for i, page\_info in zip(range(30), sorted\_page\_rank\_score\_after\_trustrank)]

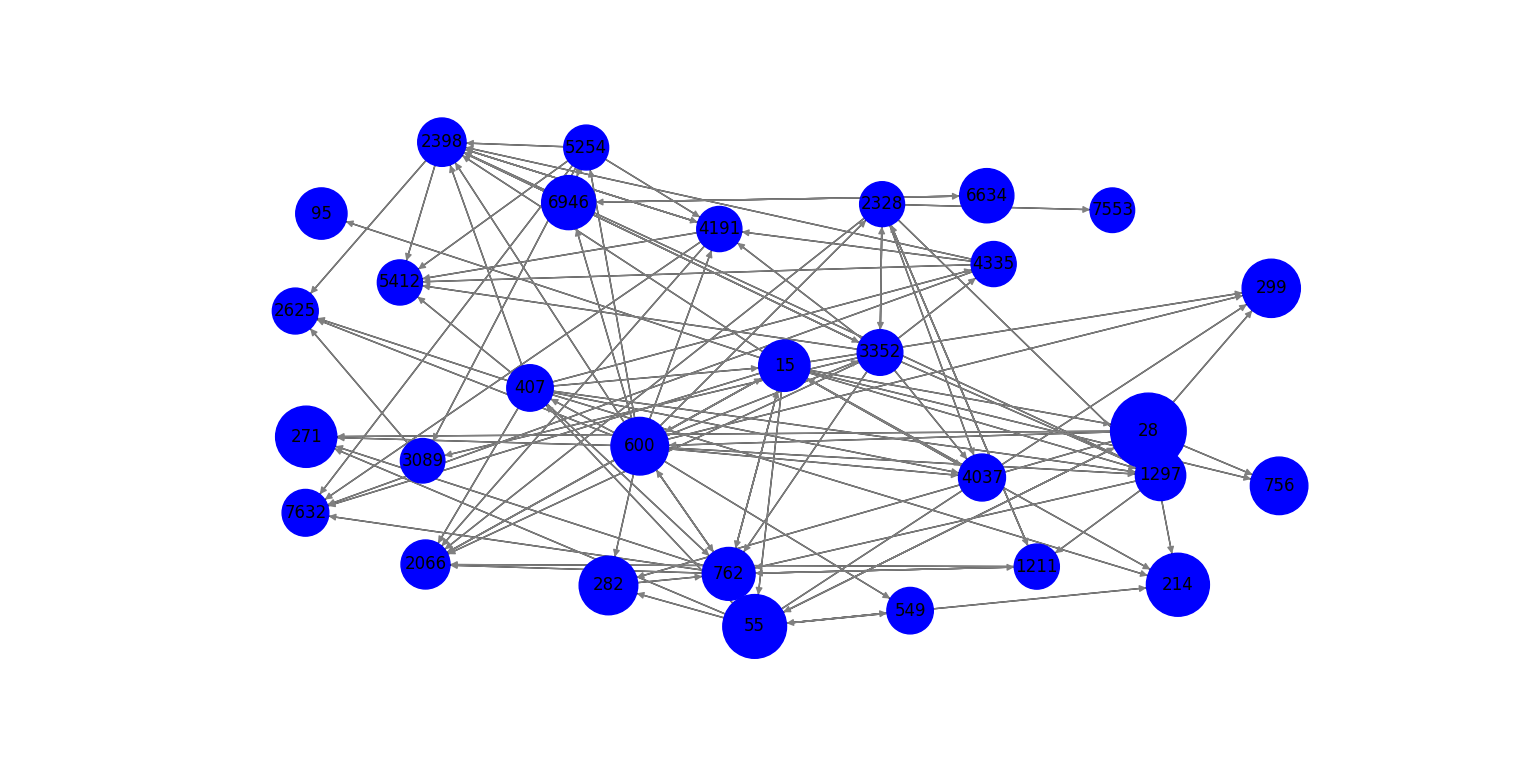
pos = nx.spring\_layout(G, k = 1, iterations = 20)

nx.draw\_networkx\_edges(G,pos)

nx.draw(G, pos, node\_size = final\_node\_size, node\_color = 'Blue', edge\_color = edge\_colors,edge\_cmap=plt.cm.Reds, with\_labels = True)

pylab.show()

Output:



Conclusion: We have successfully understand the concept of page ranking algorithm and implemented in python.