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 T_1 to T_n 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
```

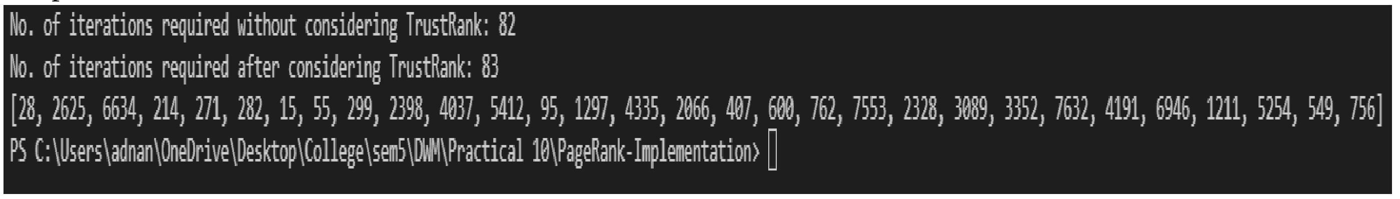
```
trusted_pages_ratio = 0.4
trusted_pages = []
maxer = 0
nodes_dict = {}
nodes = []
count = 0
beta = 0.85
# Reading form file and then saving the data to a dictionary
# of the form {NODE: [Set of nodes being pointed to by the "NODE"]}
with open("data.txt", "r") as data_file:
        for line in data_file:
                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
```

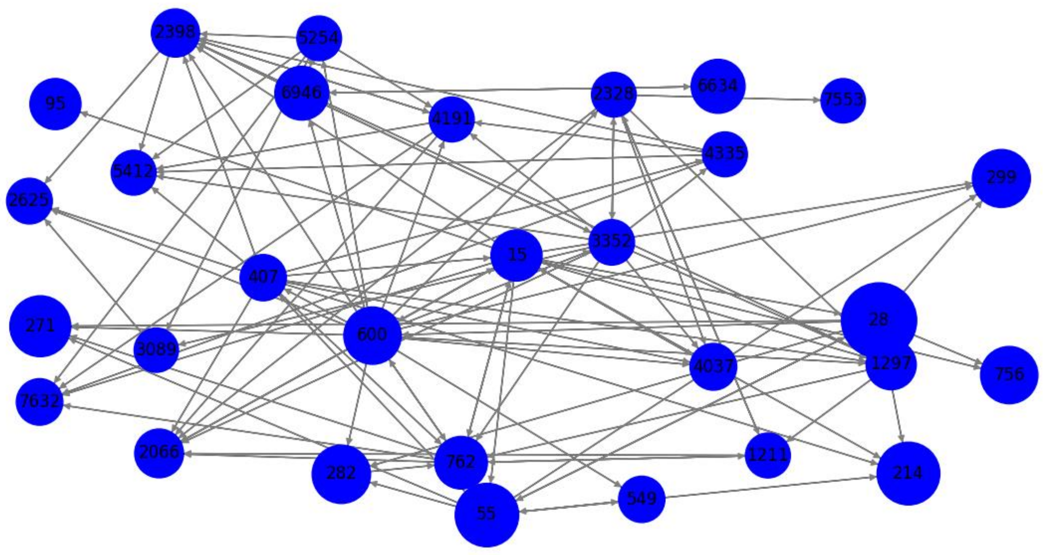
```
no_of_trusted_pages = int(trusted_pages_ratio * len(sorted_page_rank_score))
trusted_pages = [page_info[0] for i, page_info in zip(range(0, no_of_trusted_pages), sorted_page_rank_score)]
fraction = 1 / no_of_trusted_pages
# Defining initial v matrix
v = np.zeros(maxer + 1)
for i in range(1, maxer + 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:



```
No. of iterations required without considering TrustRank: 82
No. of iterations required after considering TrustRank: 83
[28, 2625, 6634, 214, 271, 282, 15, 55, 299, 2398, 4037, 5412, 95, 1297, 4335, 2066, 407, 600, 762, 7553, 2328, 3089, 3352, 7632, 4191, 6946, 1211, 5254, 549, 756]
PS C:\Users\adnan\OneDrive\Desktop\College\sem5\DWM\Practical 10\PageRank-Implementation>
```



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