Web Mining Lab - 7

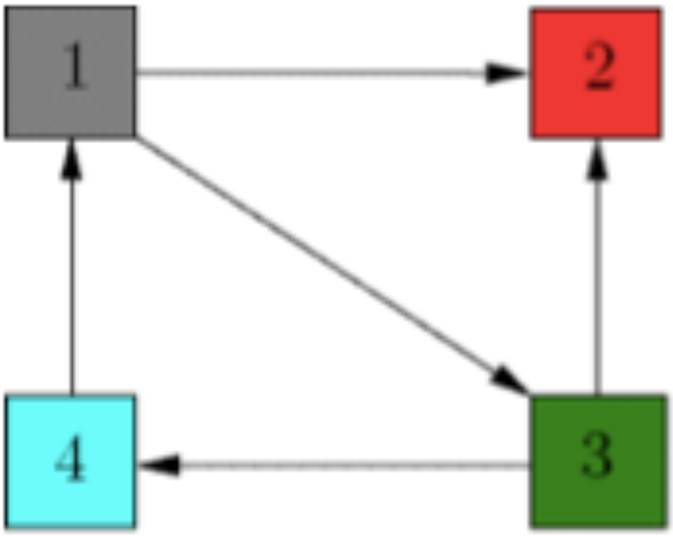
Aadhitya Swarnesh
18BCE1087

- 4 April 2021

Question - 1

Compute the Hubs and authority weights for the following graph.

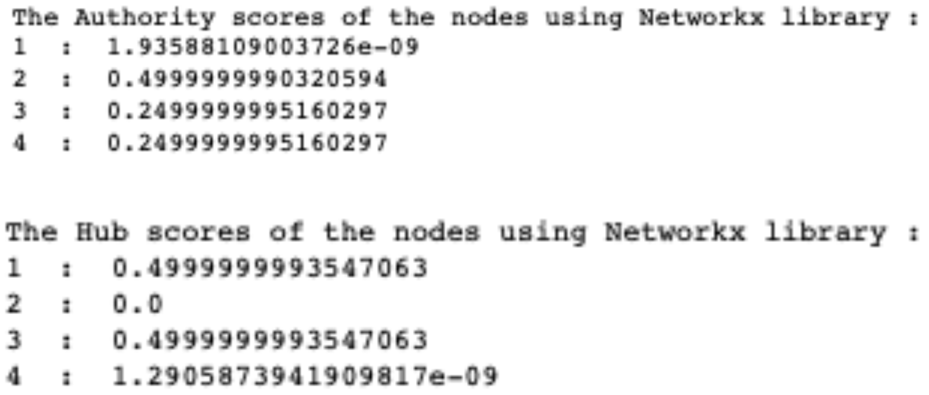
Verify the consistency of results obtained with the Custom Implementation and inbuilt HITS Algorithm function.



Before moving on to the actual question, we need to first model the graph given to us. In order to do so, we form an adjacency matrix for the graph shown above. This is available in the code section for reference.

Now to move on to the first part of the question, we will use the networkX library to model the graph, and then use the hits function available in this library to compute the authority and the hub scores of these vertices. We use a Directed Graph here because the like which connect one web page to another is just in one direction, and there is no need for a link to be in both ways.

The result obtained after following this method is as follows :



```
The Authority scores of the nodes using Networkx library :
1   :   1.93588109003726e-09
2   :   0.4999999990320594
3   :   0.2499999995160297
4   :   0.2499999995160297

The Hub scores of the nodes using Networkx library :
1   :   0.4999999993547063
2   :   0.0
3   :   0.4999999993547063
4   :   1.2905873941909817e-09
```

Now for the second part of the question, we are to validate the results obtained in the first half by implementing the actual HITS algorithm which was present inside the back box representation used above because of the networkX library. In order to do this we first find all the in-bound and the out-bound connections for all the vertices of the graph, decide on a maximum number of iterations to perform in case the values do not seem to converge.

The formula to calculate the page rank for a vertex in an iteration is as follows :

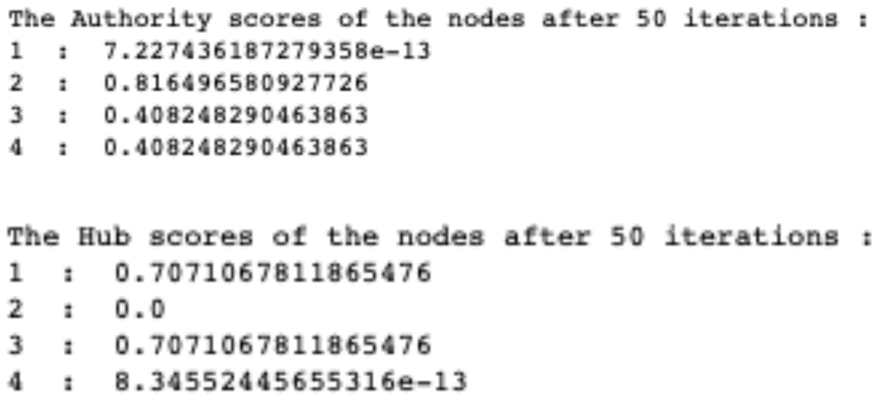
$$Authority(u) = \sum_{v \in I_u} Hub(u)$$

$$Hub(u) = \sum_{v \in O_u} Authority(u)$$

where : I_u denotes the list of all the in-bound nodes to the node **u.**
O_u denotes the list of all the out-bound nodes to the node **u.**
Authority(v) denote the Authority score of the node **v**.
Hub(v) denote the Hub score of the node **v**.

We will also be performing normalisation of the values which follows the process of dividing each value of the authority score by the square root of the sum of squares of all the authority values. A similar procedure is followed for the hub scores as well.

We use this formula for the calculating the Hub and the Authority scores, and the result obtained after the iterations is as follows :



```
The Authority scores of the nodes after 50 iterations :
1   :   7.227436187279358e-13
2   :   0.816496580927726
3   :   0.408248290463863
4   :   0.408248290463863

The Hub scores of the nodes after 50 iterations :
1   :   0.7071067811865476
2   :   0.0
3   :   0.7071067811865476
4   :   8.345524456553l6e-13
```

We can now conclude by the results obtained from both the methods that the first method was in fact valid and had produced the correct result.

The result is the following with the page with the highest Authority score in the front :

$$2 > 4 = 3 > 1$$

The result is the following with the page with the highest Hub score in the front :

$$1 = 3 > 4 > 2$$

The **code** that has delivered these results will be attached with this document below.

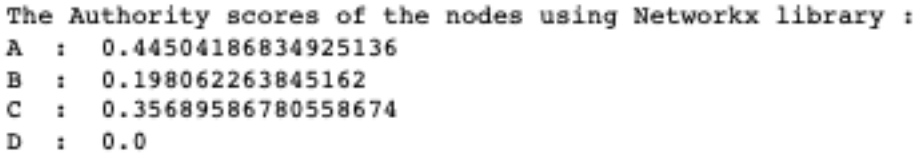
Question - 2

Assume a small universe of four web pages: A, B, C and D. Links from a page to itself, or multiple outbound links from one single page to another single page, are ignored. Page B had a link to pages C and A, page C had a link to page A, and page D had links to all three pages . Compute the Authority and Hub scores of the given web pages.

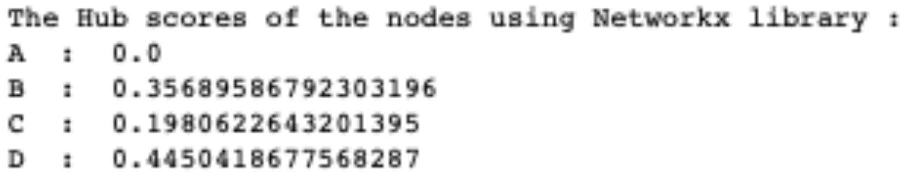
Verify the consistency of results obtained with the Custom Implementation and inbuilt HITS Algorithm function.

We have already explained in detail the whole process in the above question, and the same procedure will be followed here as well. Considering this we will now show the results of the same process.

The result obtained after following the method of the usage of **NetworkX** library is as follows :

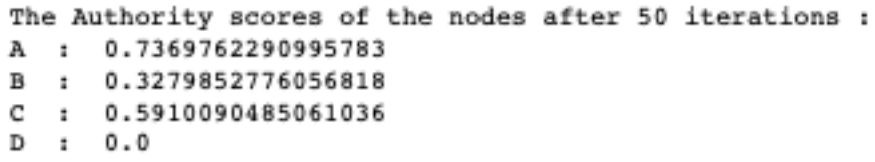


```
The Authority scores of the nodes using Networkx library :
A  :  0.44504186834925136
B  :  0.198062263845162
C  :  0.35689586780558674
D  :  0.0
```

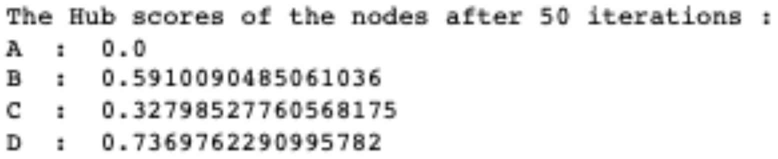


```
The Hub scores of the nodes using Networkx library :
A  :  0.0
B  :  0.35689586792303196
C  :  0.1980622643201395
D  :  0.4450418677568287
```

We also use the program made by us implementing the HITS algorithm, and the result obtained here is as follows :



```
The Authority scores of the nodes after 50 iterations :
A  :  0.7369762290995783
B  :  0.3279852776056818
C  :  0.5910090485061036
D  :  0.0
```



```
The Hub scores of the nodes after 50 iterations :
A  :  0.0
B  :  0.5910090485061036
C  :  0.32798527760568175
D  :  0.7369762290995782
```

We can now conclude by the results obtained from both the methods that the first method was in fact valid and had produced the correct result.

The result is the following with the page with the highest Authority score in the front :

$$A > C > B > D$$

The result is the following with the page with the highest Hub score in the front :

$$D > B > C > A$$

The **code** that has delivered these results will be attached with this document below

HITS Algorithm

This is a `Hyperlink Induced Topic Search` Algorithm.
We will solve two questions in this notebook, using both the `networkx` library, and also using our own custom implementation.

[Reference Link (https://www.geeksforgeeks.org/hyperlink-induced-topic-search-hits-algorithm-using-networxx-module-python/)](https://www.geeksforgeeks.org/hyperlink-induced-topic-search-hits-algorithm-using-networxx-module-python/)

In [1]:

```
# For finding square root, and other mathematical calculation
import math

# For the library based networkx implementation of the HITS algorithm
import networkx as nx

# Import the Pyplot library for plotting the created graph
from matplotlib import pyplot as plt
```

In [2]:

```
# Number of Maximum Iterations specfied

num_iterations = 50
```

Question - 1

Compute the Hubs and authority weights for the following graph.

image.png

1.1 Custom Implementation

In [3]:

```
# Names of the vertices
vertices_names_1 = ['1', '2', '3', '4']

# Number of vertices
num_vertices_1 = 4
```

In [4]:

```
# We will first form a adjacency matrix denoting the relationships shown in the image above.

adjacency_matrix_1 = [
    [0, 1, 1, 0],
    [0, 0, 0, 0],
    [0, 1, 0, 1],
    [1, 0, 0, 0]
]
```

In [5]:

```
# Convert the adjacency matrix into any other form for the easy access of the data.

# Store the out-bound vertices for each vertex
outbound_vertices_1 = [[] for _ in range(num_vertices_1)]

for i in range(num_vertices_1) :
    for j in range(num_vertices_1) :
        if adjacency_matrix_1[i][j] == 1 :
            outbound_vertices_1[i].append(j)

outbound_vertices_1
```

Out[5]:

```
[[1, 2], [], [1, 3], [0]]
```

In [6]:

```
# Store the in-bound vertices for each vertex
inbound_vertices_1 = [[] for _ in range(num_vertices_1)]

for i in range(num_vertices_1) :
    for j in range(num_vertices_1) :
        if adjacency_matrix_1[j][i] == 1 :
            inbound_vertices_1[i].append(j)

inbound_vertices_1
```

Out[6]:

```
[[3], [0, 2], [0], [2]]
```

In [43]:

```
# We will initialize the `authority` and the `hub` scores.

authority_scores_1 = [1] * num_vertices_1
hub_scores_1 = [1] * num_vertices_1
```

In [44]:

```
# Implement the algorithm for the specfied number of iterations

for itr in range(num_iterations) :

    # Copy the current authority, and hub scores
    old_authority_scores = authority_scores_1[:]
    old_hub_scores = hub_scores_1[:]

    # Update the authority and hub scores
    for i in range(num_vertices_1) :
        authority_scores_1[i] = sum([old_hub_scores[j] for j in inbound_vertices_1[i]])
        hub_scores_1[i] = sum([old_authority_scores[j] for j in outbound_vertices_1[i]])

    # Normalize the values
    a_normal = math.sqrt(sum([i**2 for i in authority_scores_1]))
    h_normal = math.sqrt(sum([i**2 for i in hub_scores_1]))
    for i in range(num_vertices_1) :
        authority_scores_1[i] = authority_scores_1[i] / a_normal
        hub_scores_1[i] = hub_scores_1[i] / h_normal
```

In [45]:

```
# Print the Authority scores

print(f"The Authority scores of the nodes after {num_iterations} iterations : ")
for i in range(num_vertices_1) :
    print(vertices_names_1[i], " : ", authority_scores_1[i])
```

```
The Authority scores of the nodes after 50 iterations :
1  :  7.227436187279358e-13
2  :  0.816496580927726
3  :  0.408248290463863
4  :  0.408248290463863
```

In [46]:

```
# Print the Hub scores

print(f"The Hub scores of the nodes after {num_iterations} iterations : ")
for i in range(num_vertices_1) :
    print(vertices_names_1[i], " : ", hub_scores_1[i])
```

```
The Hub scores of the nodes after 50 iterations :
1  :  0.7071067811865476
2  :  0.0
3  :  0.7071067811865476
4  :  8.34552445655316e-13
```

1.2 Networkx Implementation

In [16]:

```
# Create an instance of the Graph class
# We are using a Directed graph due to the nature of our problem.

graph_1 = nx.DiGraph()
```

In [18]:

```
# Load the nodes into the graph

graph_1.add_nodes_from(vertices_names_1)
```

In [19]:

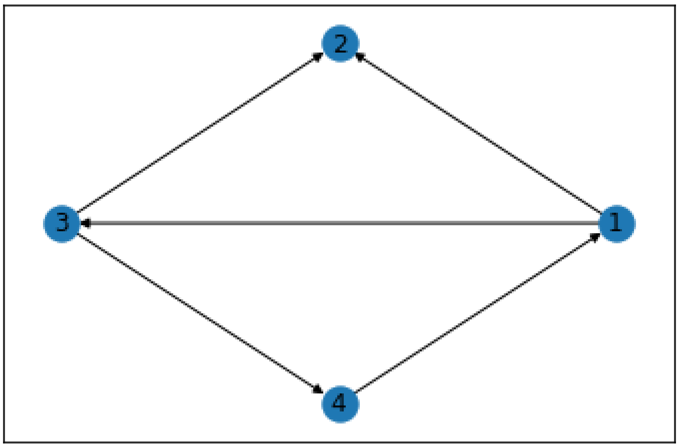
```
# Add the edges from the adjacency matrix

for i in range(num_vertices_1) :
    for j in range(num_vertices_1) :
        if adjacency_matrix_1[i][j] == 1 :
            graph_1.add_edge(vertices_names_1[i], vertices_names_1[j])
```

In [20]:

```
# Draw generated graph

nx.draw_networkx(graph_1, pos=nx.circular_layout(graph_1), arrows=True, with_labels=True)
plt.show()
```



In [23]:

```
# Compute the scores

hub_scores_networkx_1, authority_scores_networkx_1 = nx.hits(graph_1, normalized = True)
```

In [49]:

```
# Print the Authority scores

print("The Authority scores of the nodes using Networkx library : ")
for k, v in authority_scores_networkx_1.items() :
    print(k, " : ", v)
```

```
The Authority scores of the nodes using Networkx library :
1  :  1.93588109003726e-09
2  :  0.4999999990320594
3  :  0.2499999995160297
4  :  0.2499999995160297
```

In [50]:

```
# Print the Hub scores

print("The Hub scores of the nodes using Networkx library : ")
for k, v in hub_scores_networkx_1.items() :
    print(k, " : ", v)
```

```
The Hub scores of the nodes using Networkx library :
1  :  0.4999999993547063
2  :  0.0
3  :  0.4999999993547063
4  :  1.2905873941909817e-09
```

1.3 Result

We can notice that both these methods give the authority and hub scores of the pages in the same order.

In [ ]:

```
```

Question - 2

Assume a small universe of four web pages: `A, B, C, D`. Links from a page to itself, or multiple outbound links from one single page to another single page, are ignored.
The following is the configaration of the nodes :

- Page B had a link to pages C and A
- Page C had a link to page A
- Page D had links to all three pages

Compute the `Authority` and `Hub` scores of the given web pages.

2.1 Custom Implementation

In [3]:

```
# Names of the vertices
vertices_names_2 = ['A', 'B', 'C', 'D']

# Number of vertices
num_vertices_2 = 4
```

In [4]:

```
# We will first form a adjacency matrix denoting the relationships shown in the
 image above.

adjacency_matrix_2 = [
    [0, 0, 0, 0],
    [1, 0, 1, 0],
    [1, 0, 0, 0],
    [1, 1, 1, 0]
]
```

In [5]:

```
# Convert the adjacency matrix into any other form for the easy access of the da
ta.

# Store the out-bound vertices for each vertex
outbound_vertices_2 = [[] for _ in range(num_vertices_2)]

for i in range(num_vertices_2) :
    for j in range(num_vertices_2) :
        if adjacency_matrix_2[i][j] == 1 :
            outbound_vertices_2[i].append(j)

outbound_vertices_2
```

Out[5]:

```
[[], [0, 2], [0], [0, 1, 2]]
```

In [6]:

```
# Store the in-bound vertices for each vertex
inbound_vertices_2 = [[] for _ in range(num_vertices_2)]

for i in range(num_vertices_2) :
    for j in range(num_vertices_2) :
        if adjacency_matrix_2[j][i] == 1 :
            inbound_vertices_2[i].append(j)

inbound_vertices_2
```

Out[6]:

```
[[1, 2, 3], [3], [1, 3], []]
```

In [7]:

```
# We will initialize the `authority` and the `hub` scores.

authority_scores_2 = [1] * num_vertices_2
hub_scores_2 = [1] * num_vertices_2
```

In [8]:

```
# Implement the algorithm for the specfied number of iterations

for itr in range(num_iterations) :

    # Copy the current authority, and hub scores
    old_authority_scores = authority_scores_2[:]
    old_hub_scores = hub_scores_2[:]

    # Update the authority and hub scores
    for i in range(num_vertices_2) :
        authority_scores_2[i] = sum([old_hub_scores[j] for j in inbound_vertices
_2[i]])
        hub_scores_2[i] = sum([old_authority_scores[j] for j in outbound_vertice
s_2[i]])

    # Normalize the values
    a_normal = math.sqrt(sum([i**2 for i in authority_scores_2]))
    h_normal = math.sqrt(sum([i**2 for i in hub_scores_2]))
    for i in range(num_vertices_2) :
        authority_scores_2[i] = authority_scores_2[i] / a_normal
        hub_scores_2[i] = hub_scores_2[i] / h_normal
```

In [9]:

```
# Print the Authority scores

print(f"The Authority scores of the nodes after {num_iterations} iterations : ")
for i in range(num_vertices_2) :
    print(vertices_names_2[i], " : ", authority_scores_2[i])
```

```
The Authority scores of the nodes after 50 iterations :
A  :  0.7369762290995783
B  :  0.3279852776056818
C  :  0.5910090485061036
D  :  0.0
```

In [10]:

```
# Print the Hub scores

print(f"The Hub scores of the nodes after {num_iterations} iterations : ")
for i in range(num_vertices_2) :
    print(vertices_names_2[i], " : ", hub_scores_2[i])
```

```
The Hub scores of the nodes after 50 iterations :
A  :  0.0
B  :  0.5910090485061036
C  :  0.32798527760568175
D  :  0.7369762290995782
```

2.2 Networkx Implementation

In [11]:

```
# Create an instance of the Graph class
# We are using a Directed graph due to the nature of our problem.

graph_2 = nx.DiGraph()
```

In [12]:

```
# Load the nodes into the graph

graph_2.add_nodes_from(vertices_names_2)
```

In [13]:

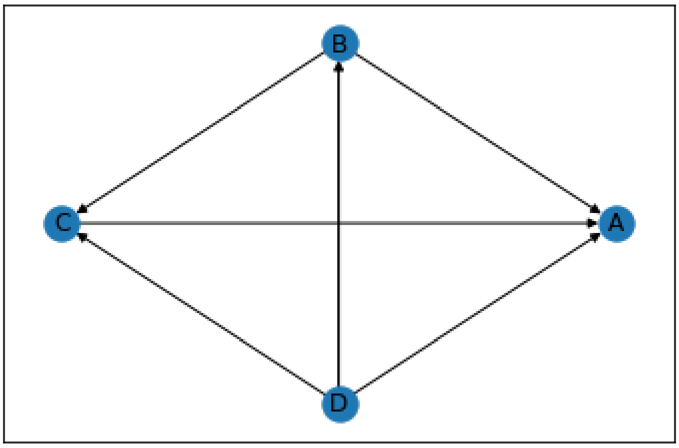
```
# Add the edges from the adjacency matrix

for i in range(num_vertices_2) :
    for j in range(num_vertices_2) :
        if adjacency_matrix_2[i][j] == 1 :
            graph_2.add_edge(vertices_names_2[i], vertices_names_2[j])
```

In [14]:

```
# Draw generated graph

nx.draw_networkx(graph_2, pos=nx.circular_layout(graph_2), arrows=True, with_labels=True)
plt.show()
```



In [15]:

```
# Compute the scores

hub_scores_networkx_2, authority_scores_networkx_2 = nx.hits(graph_2, normalized = True)
```

In [16]:

```
# Print the Authority scores

print("The Authority scores of the nodes using Networkx library : ")
for k, v in authority_scores_networkx_2.items() :
    print(k, " : ", v)
```

```
The Authority scores of the nodes using Networkx library :
A  :  0.44504186834925136
B  :  0.198062263845162
C  :  0.35689586780558674
D  :  0.0
```

In [17]:

```
# Print the Hub scores

print("The Hub scores of the nodes using Networkx library : ")
for k, v in hub_scores_networkx_2.items() :
    print(k, " : ", v)
```

```
The Hub scores of the nodes using Networkx library :
A  :  0.0
B  :  0.35689586792303196
C  :  0.1980622643201395
D  :  0.4450418677568287
```

2.3 Result

We can notice that both these methods give the authority and hub scores of the pages in the same order.

In [ ]:

```

```