WEB MINING LAB

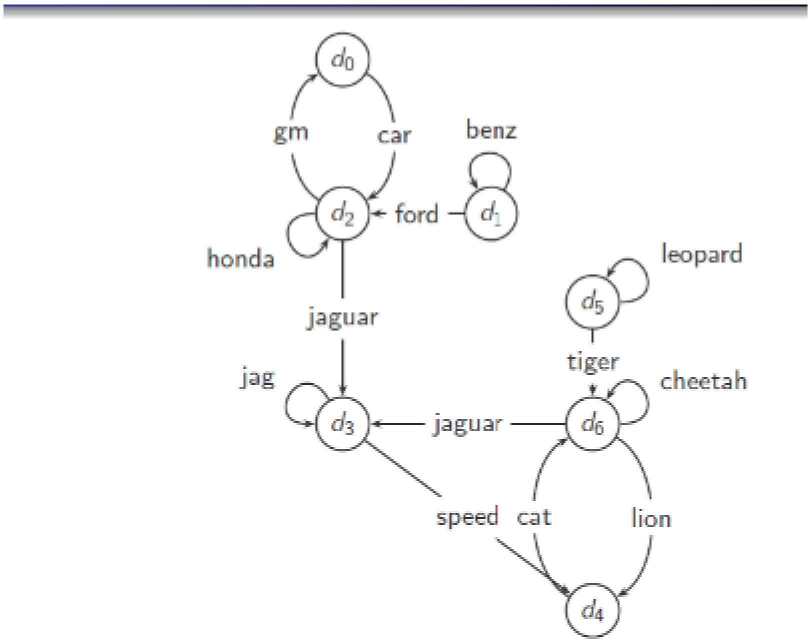
Faculty: Dr.Sridhar.R

LAB5
DATE: 28TH SEPT 2021

VIT CHENNAI

BELIDA KARTHIK
19BCE1446

Aim: to implement and understand hits algorithm



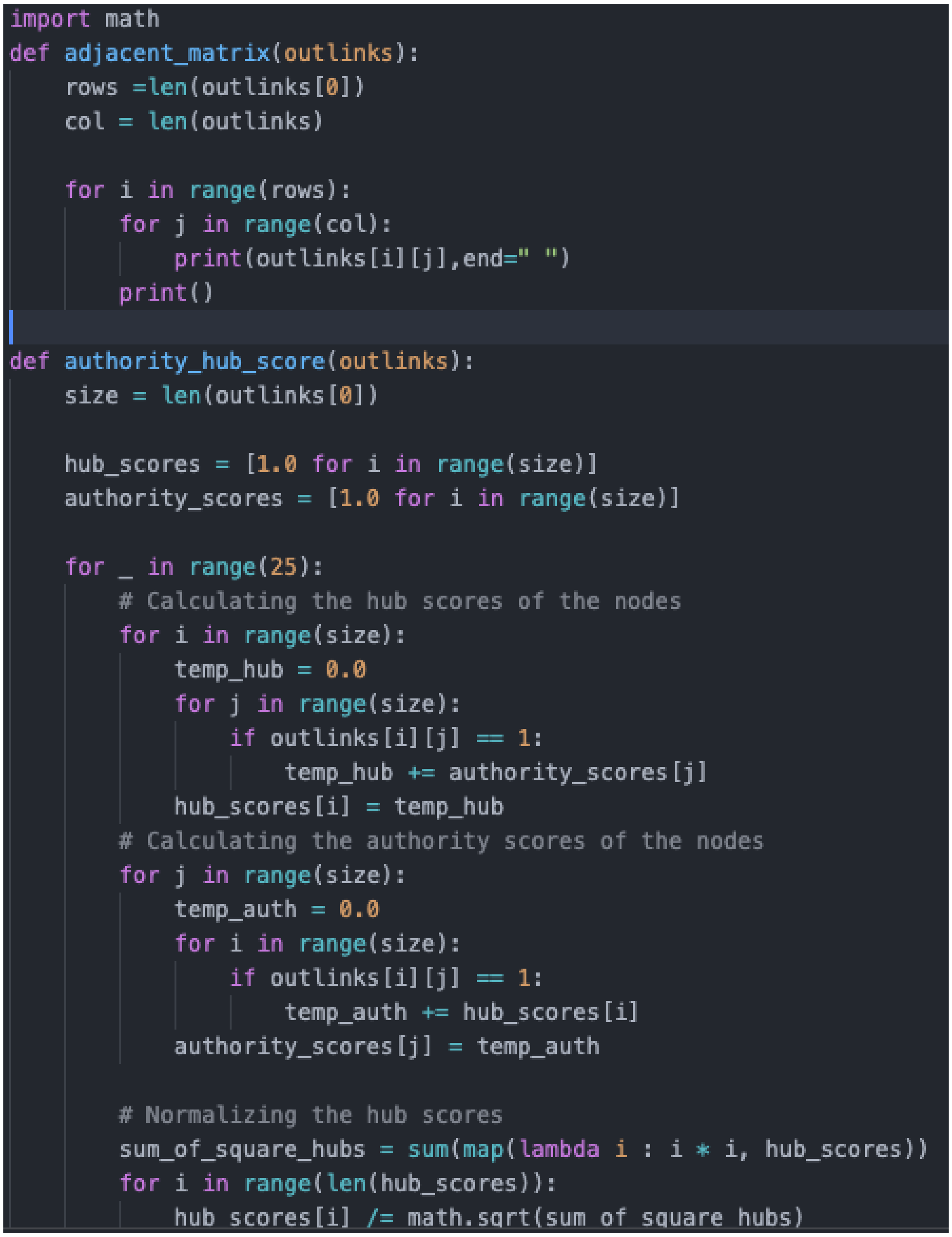
1. Form the adjacency graph of this **IGNORING SELF LOOPS with the following principles**

a. dx-dy entry is 1 if there is a link

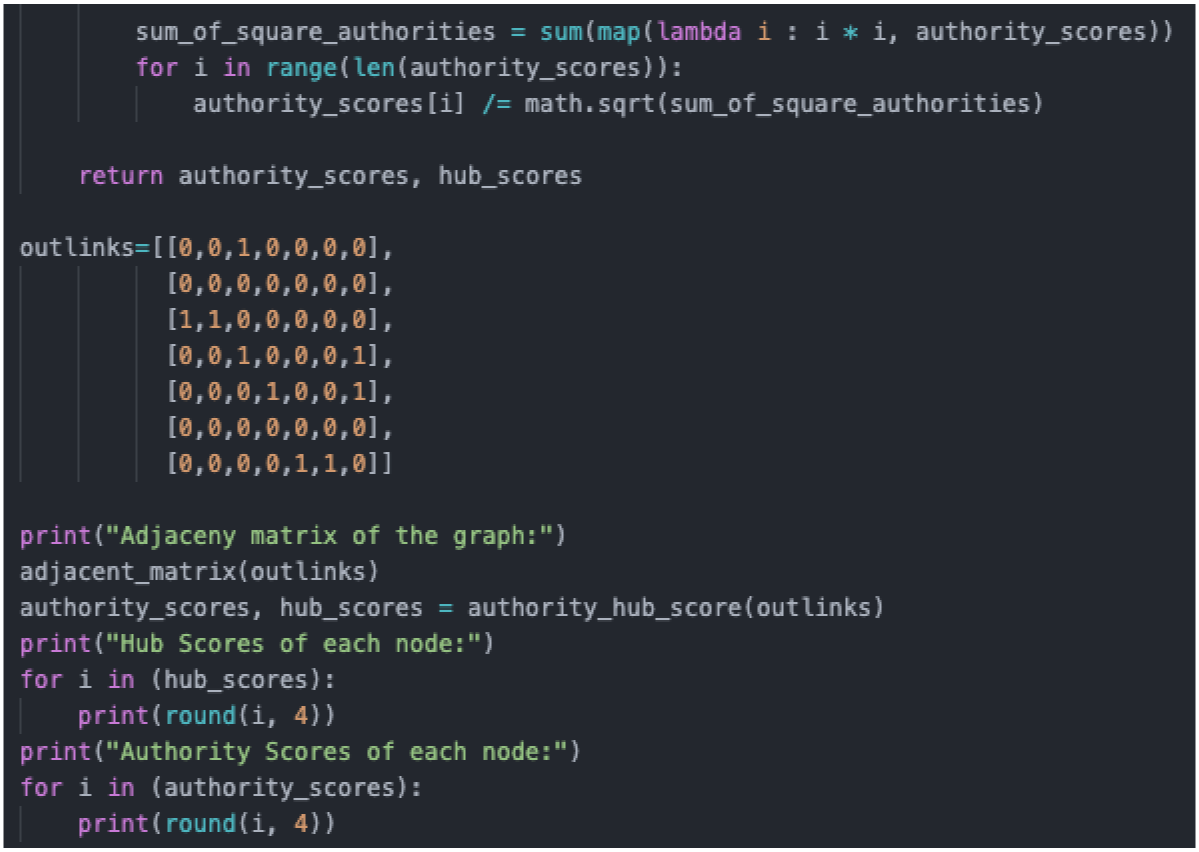
b. dx-dy entry is 0 if there is no link

Calculate the Hub score and Authority score for this graph **IGNORING SELF LOOPS** by writing a program in python**.** Perform 25 iterations and print out the final values of Hub score and authority score for all nodes.

Code:

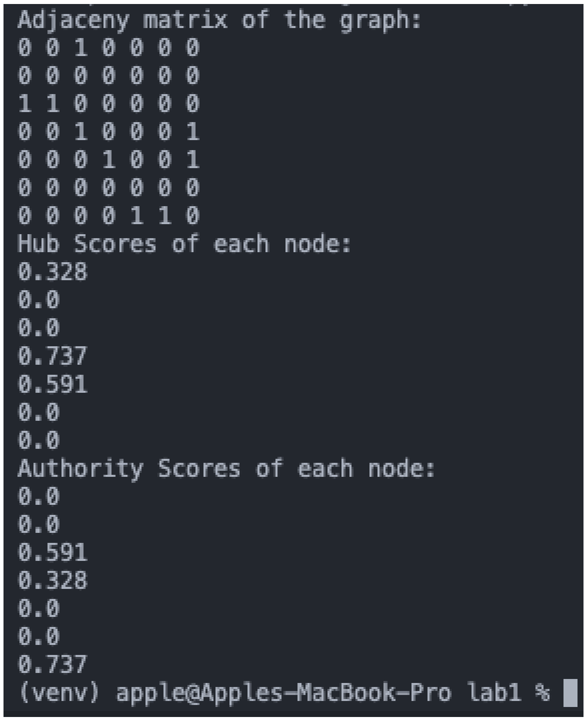


```python
import math
def adjacent_matrix(outlinks):
    rows =len(outlinks[0])
    col = len(outlinks)

    for i in range(rows):
        for j in range(col):
            print(outlinks[i][j],end=" ")
        print()

def authority_hub_score(outlinks):
    size = len(outlinks[0])

    hub_scores = [1.0 for i in range(size)]
    authority_scores = [1.0 for i in range(size)]

    for _ in range(25):
        # Calculating the hub scores of the nodes
        for i in range(size):
            temp_hub = 0.0
            for j in range(size):
                if outlinks[i][j] == 1:
                    temp_hub += authority_scores[j]
            hub_scores[i] = temp_hub
        # Calculating the authority scores of the nodes
        for j in range(size):
            temp_auth = 0.0
            for i in range(size):
                if outlinks[i][j] == 1:
                    temp_auth += hub_scores[i]
            authority_scores[j] = temp_auth

        # Normalizing the hub scores
        sum_of_square_hubs = sum(map(lambda i : i * i, hub_scores))
        for i in range(len(hub_scores)):
            hub_scores[i] /= math.sqrt(sum_of_square_hubs)
```



```
        sum_of_square_authorities = sum(map(lambda i : i * i, authority_scores))
        for i in range(len(authority_scores)):
            authority_scores[i] /= math.sqrt(sum_of_square_authorities)

    return authority_scores, hub_scores

outlinks=[[0,0,1,0,0,0,0],
          [0,0,0,0,0,0,0],
          [1,1,0,0,0,0,0],
          [0,0,1,0,0,0,1],
          [0,0,0,1,0,0,1],
          [0,0,0,0,0,0,0],
          [0,0,0,0,1,1,0]]

print("Adjaceny matrix of the graph:")
adjacent_matrix(outlinks)
authority_scores, hub_scores = authority_hub_score(outlinks)
print("Hub Scores of each node:")
for i in (hub_scores):
    print(round(i, 4))
print("Authority Scores of each node:")
for i in (authority_scores):
    print(round(i, 4))
```

OutPut:



```
Adjaceny matrix of the graph:
0 0 1 0 0 0 0
0 0 0 0 0 0 0
1 1 0 0 0 0 0
0 0 1 0 0 0 1
0 0 0 1 0 0 1
0 0 0 0 0 0 0
0 0 0 0 1 1 0
Hub Scores of each node:
0.328
0.0
0.0
0.737
0.591
0.0
0.0
Authority Scores of each node:
0.0
0.0
0.591
0.328
0.0
0.0
0.737
(venv) apple@Apples-MacBook-Pro lab1 %
```

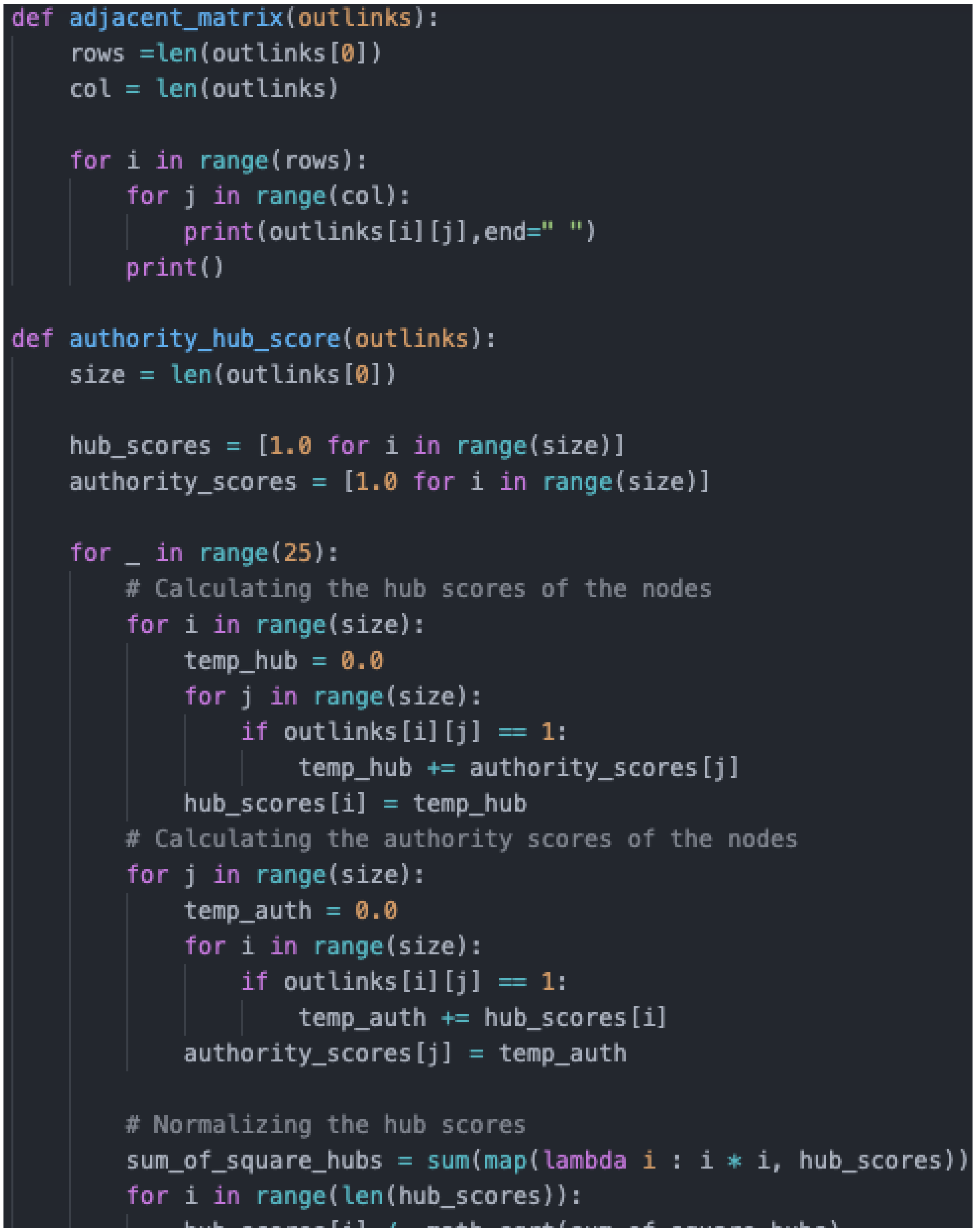
2. Form the adjacency graph of this INCLUDING SELF LOOPS with the following principles.

a.dx-dy entry is 1 if there is a link

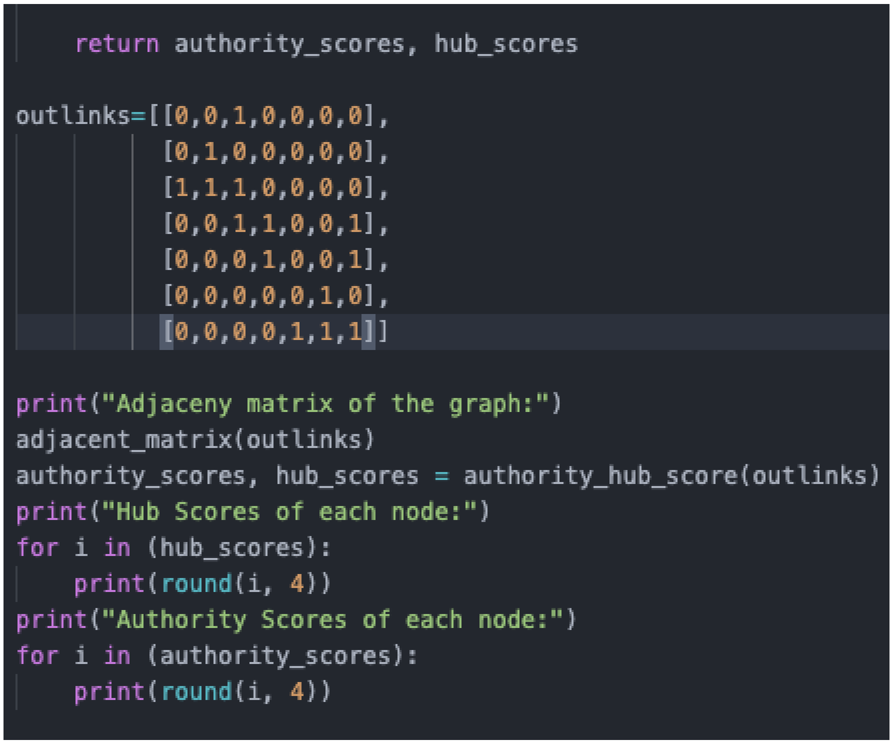
b.dx-dy entry is 0 if there is no link

c.Calculate the Hub score and Authority score for this graph INCLUDING SELF LOOPS by writing a program in python. Perform 25 iterations and print out the final values of Hub score and authority score for all nodes.

Code:

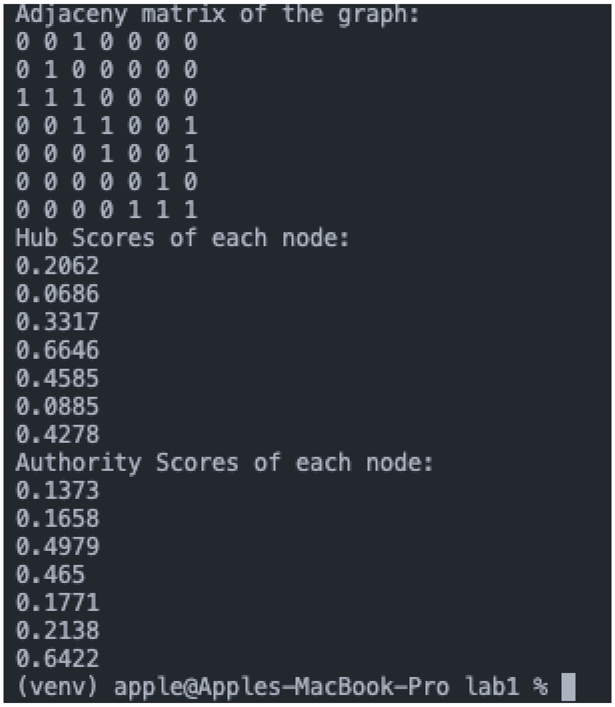


```
def adjacent_matrix(outlinks):
    rows =len(outlinks[0])
    col = len(outlinks)

    for i in range(rows):
        for j in range(col):
            print(outlinks[i][j],end=" ")
        print()

def authority_hub_score(outlinks):
    size = len(outlinks[0])

    hub_scores = [1.0 for i in range(size)]
    authority_scores = [1.0 for i in range(size)]

    for _ in range(25):
        # Calculating the hub scores of the nodes
        for i in range(size):
            temp_hub = 0.0
            for j in range(size):
                if outlinks[i][j] == 1:
                    temp_hub += authority_scores[j]
            hub_scores[i] = temp_hub
        # Calculating the authority scores of the nodes
        for j in range(size):
            temp_auth = 0.0
            for i in range(size):
                if outlinks[i][j] == 1:
                    temp_auth += hub_scores[i]
            authority_scores[j] = temp_auth

        # Normalizing the hub scores
        sum_of_square_hubs = sum(map(lambda i : i * i, hub_scores))
        for i in range(len(hub_scores)):
            hub_scores[i] /= math.sqrt(sum_of_square_hubs)
```



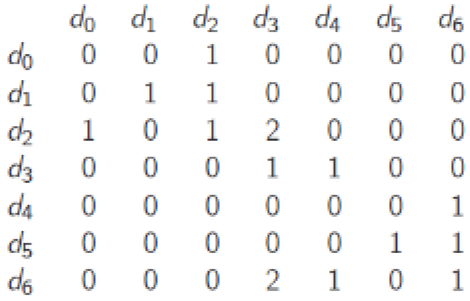
```
    return authority_scores, hub_scores

outlinks=[[0,0,1,0,0,0,0],
          [0,1,0,0,0,0,0],
          [1,1,1,0,0,0,0],
          [0,0,1,1,0,0,1],
          [0,0,0,1,0,0,1],
          [0,0,0,0,0,1,0],
          [0,0,0,0,1,1,1]]

print("Adjaceny matrix of the graph:")
adjacent_matrix(outlinks)
authority_scores, hub_scores = authority_hub_score(outlinks)
print("Hub Scores of each node:")
for i in (hub_scores):
    print(round(i, 4))
print("Authority Scores of each node:")
for i in (authority_scores):
    print(round(i, 4))
```

Output:



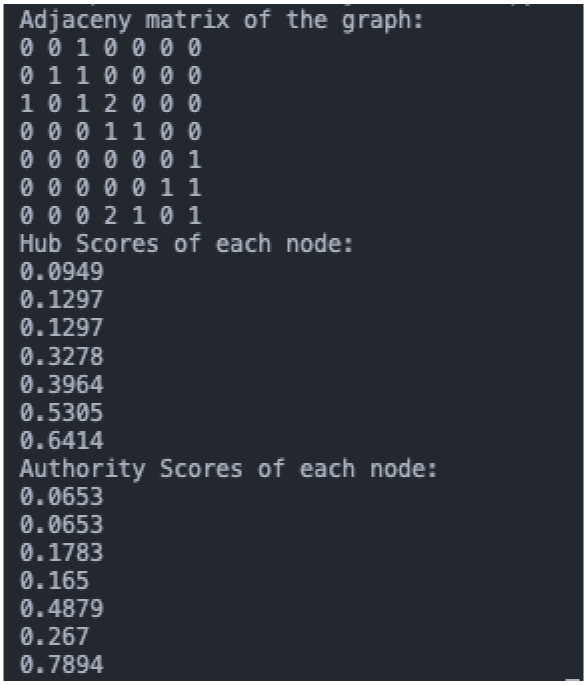
```
Adjaceny matrix of the graph:
0 0 1 0 0 0 0
0 1 0 0 0 0 0
1 1 1 0 0 0 0
0 0 1 1 0 0 1
0 0 0 1 0 0 1
0 0 0 0 0 1 0
0 0 0 0 1 1 1
Hub Scores of each node:
0.2062
0.0686
0.3317
0.6646
0.4585
0.0885
0.4278
Authority Scores of each node:
0.1373
0.1658
0.4979
0.465
0.1771
0.2138
0.6422
(venv) apple@Apples-MacBook-Pro lab1 %
```

3. Calculate the Hub score and Authority score for this graph IGNORING SELF LOOPS by writing a program in python. Perform 25 iterations and print out the final values of Hub score and authority score for all nodes.



$$\begin{array}{c|ccccccc}
 & d_0 & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 \\
d_0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
d_1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
d_2 & 1 & 0 & 1 & 2 & 0 & 0 & 0 \\
d_3 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
d_4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
d_5 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
d_6 & 0 & 0 & 0 & 2 & 1 & 0 & 1
\end{array}$$

Same code :(change adjacency matrix)



```
Adjaceny matrix of the graph:
0 0 1 0 0 0 0
0 1 1 0 0 0 0
1 0 1 2 0 0 0
0 0 0 1 1 0 0
0 0 0 0 0 0 1
0 0 0 0 0 1 1
0 0 0 2 1 0 1
Hub Scores of each node:
0.0949
0.1297
0.1297
0.3278
0.3964
0.5305
0.6414
Authority Scores of each node:
0.0653
0.0653
0.1783
0.165
0.4879
0.267
0.7894
```