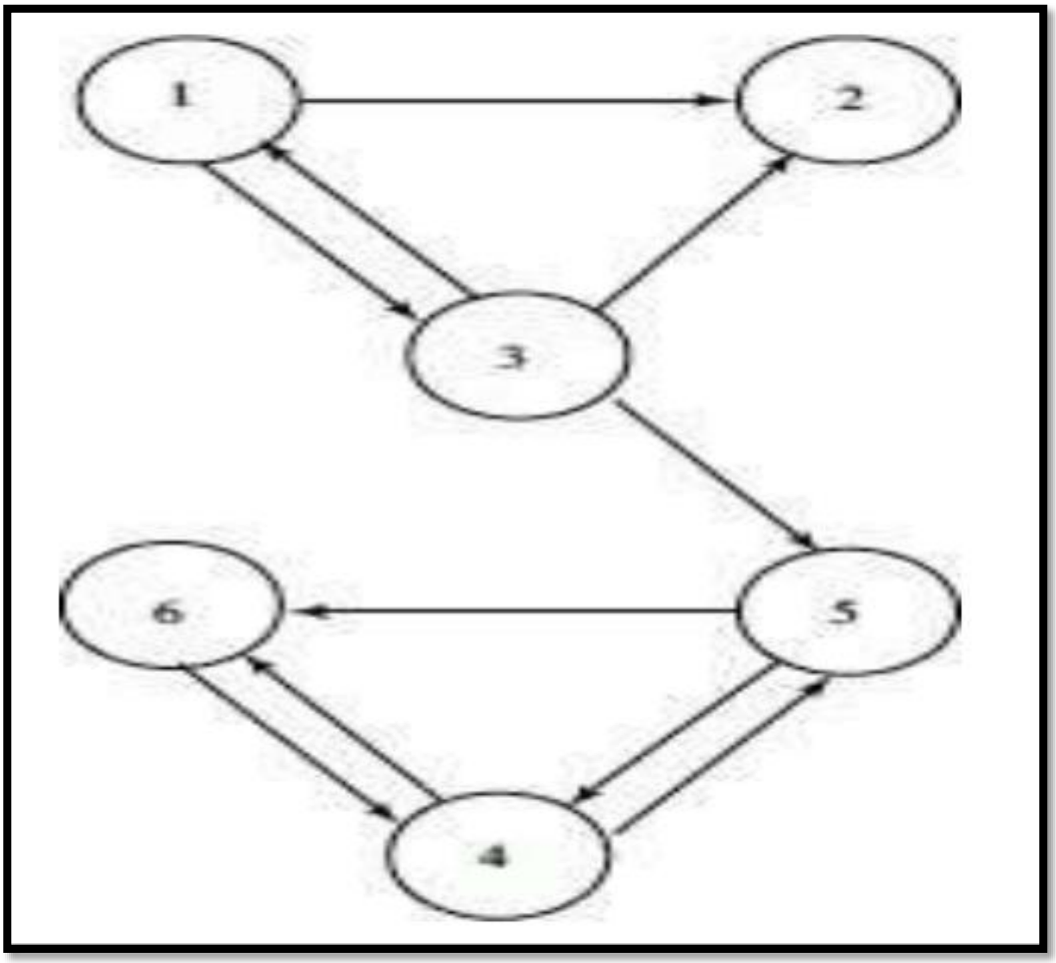
Harshit Mishra (19BCE0799)

CSE-3024 WEB MINING
LAB ASSIGNMENT 7

Aim: Write a python program to find the ranks for the given graph.



Perform **7 iteration** and print the final iteration value only.

Procedure:

- Firstly, we import the necessary libraries of numpy, scipy and sparse.
- Then write a compute page rank function, which takes in three input parameters, namely, links, damping factor and number of iterations.
- We initialize the damping factor to a standard 0.85 value and number of iterations to 7 as mentioned in question.
- Then we use that function to compute page rank of each page in our network.

Harshit Mishra (19BCE0799)

Code:

```
#Importing libraries
import scipy
from scipy import sparse
import numpy


#Computing Page Rank function
def computePageRank(links, c=0.85, iteration=7):
    count = 0
    ones = numpy.ones(len(links))
    sources = [x[0] for x in links]
    targets = [x[1] for x in links]
    n = max(max(sources), max(targets))+1
    HT = sparse.coo_matrix((ones, (targets, sources)), shape=(n,n))
    num_outlinks = numpy.array(HT.sum(axis=0)).flatten()
    HT.data/=num_outlinks[sources]
    d_indices = numpy.where(num_outlinks == 0)[0]
    r = numpy.ones(n)/n
    while True:
        previous_r = r
        r = c * (HT * r + sum(r[d_indices])/n) + (1.0 - c)/n
        #r.sum() ≈ 1 but prevent errors from adding up.
        r /= r.sum()
        count = count+1
        if(count >iteration):
            #if scipy.absolute(r - previous_r).sum() < epsilon:
            return r


print(computePageRank([(0,1), (0,2), (2,0),(2, 1),(2, 4),(3, 4),(3,5),(4,3),(4,5), (5,3) ]))
```

Code Snippet and Outputs:

```
In [1]: #Importing libraries
        import scipy
        from scipy import sparse
        import numpy
```

Here we are importing our libraries. We import scipy, numpy and sparse from scipy.

```
In [2]: #Computing Page Rank function
        def computePageRank(links, c=0.85, iteration=7):
            count = 0
            ones = numpy.ones(len(links))
            sources = [x[0] for x in links]
            targets = [x[1] for x in links]
            n = max(max(sources), max(targets))+1
            HT = sparse.coo_matrix((ones, (targets, sources)), shape=(n,n))
            num_outlinks = numpy.array(HT.sum(axis=0)).flatten()
            HT.data/=num_outlinks[sources]
            d_indices = numpy.where(num_outlinks == 0)[0]
            r = numpy.ones(n)/n
            while True:
                previous_r = r
                r = c * (HT * r + sum(r[d_indices])/n) + (1.0 - c)/n
                #r.sum() ≈ 1 but prevent errors from adding up.
                r /= r.sum()
                count = count+1
                if(count >iteration):
                    #if scipy.absolute(r - previous_r).sum() < epsilon:
                    return r
```

This is our computePageRank function and it returns us a list of page ranks of each page in our network. It takes 3 parameters as input, them being, links of nodes in our network, damping factor and number of iterations. We initialize the damping factor to be a standard of 0.85 and number of iterations to 7 as mentioned in our question.

```
In [3]: print(computePageRank([(0,1), (0,2), (2,0),(2, 1),(2, 4),(3, 4),(3,5),(4,3),(4

        [0.05276657 0.07551812 0.05864201 0.34644978 0.19951605 0.26710748]
```

Here we have computed the page rank of each page in our network. Since we didn't have a 0 node, but counting in python starts from 0, we have renamed each node in our question. They are decremented 1 each.

1 → 0

2 → 1

3 → 2

4 → 3

5 → 4

6 → 5

Results:

Page Ranks:

```
[0.05276657 0.07551812 0.05864201 0.34644978 0.19951605 0.26710748]
```

The page rank of each node in our network is as follows:

Node 1 → 0.05276657

Node 2 → 0.07551812

Node 3 → 0.05864201

Node 4 → 0.34644978

Node 5 → 0.19951605

Node 6 → 0.26710748

Conclusion and Inference:

Sum of page rank of each node in our network is 1.

The nodes in decreasing order of page ranks are:

Node 4 > Node 6 > Node 5 > Node 2 > Node 3 > Node 1