ASSIGNMENT 2

CS F469 Information Retrieval

Aryan Chaubal (2019A7PS0130H)

Samarth Jain (2019A7PS0179H)

Prathamesh Jadhav (2019A7PS0084H)

PART A: Implementation of the PageRank Algorithm

The PageRank algorithm was implemented using Python and Numpy. We created the probability transition matrix both with and without random teleportations.

In case of random teleportations, we used the probability of teleportations as 0.1.

Functions Used

Generating the probability transition matrix

```
generate_probability_transition_matrix(nodes_count, edges, use_random_teleportation=False)
```

This function generates the probability transition matrix using a list of edge tuples and the total number of nodes. It also takes in an optional flag to define the random teleportation behavior. By default, random teleportation is turned off.

The function follows the following steps:

1. We initialize a matrix of shape (node_count, node_count) with zeros.
2. For every edge from node a to b, we set matrix[a][b] = 1.
3. Next, we divide each 1 by the number of 1s in that row. With this, each row will have a sum of 1. In case we do not need random teleportation, we return the matrix here itself.
4. We multiply the matrix by (1 - α). Here α is the probability of random teleportation = 0.1
5. Next, we fill all the zeros with α / n where n is the number of zeros in that particular row. We return the final matrix.

Obtaining the left principal eigenvector (Using a Numpy function)

```
get_left_principal_eigenvector(probability_transition_matrix)
```

This function returns the left principal eigenvector that has been obtained using the `np.linealg.eig()` function. These eigenvectors also need to be sorted according to their eigenvalues and need to be normalized to obtain the principal left eigenvector.

Obtaining the left principal eigenvector (Using Power Iteration Method)

`get_left_principal_eigenvector_power_iteration(probability_transition_matrix, nodes_count)`

This function calculates the left principal eigenvector using the Power Iteration method. An initial vector is initialized with 1 / nodes_count for every element. Then we keep multiplying the `probability_transition_matrix` till the vector converges to our resultant value and then we return this value.

Data Structures used

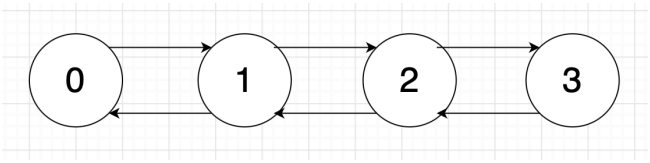
Probability Transition Matrix

This is just a matrix where an element at (i, j) indicates the probability of a transition from webpage i to webpage j. It is stored as a numpy array.

Results

The following are the probability scores obtained for the test cases given:

Test Case 1:



Result with random teleportation:

Probability vector (1, 2, 3, 4 respectively):

Probabilities using Numpy Function:

[[0.1744186 0.3255814 0.3255814 0.1744186]]

Probabilities using Power Iteration:

[[0.1744186 0.3255814 0.3255814 0.1744186]]

Result without random teleportation:

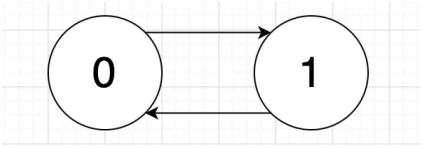
Probabilities using Numpy Function:

[[0.16666667 0.33333333 0.33333333 0.16666667]]

Probabilities using Power Iteration:

[[0.16666667 0.33333333 0.33333333 0.16666667]]

Test Case 2:



Result with random teleportation:

Probabilities using Numpy Function:

[[0.5 0.5]]

Probabilities using Power Iteration:

[[0.5 0.5]]

Result without random teleportation:

Probabilities using Numpy Function:

[[0.5 0.5]]

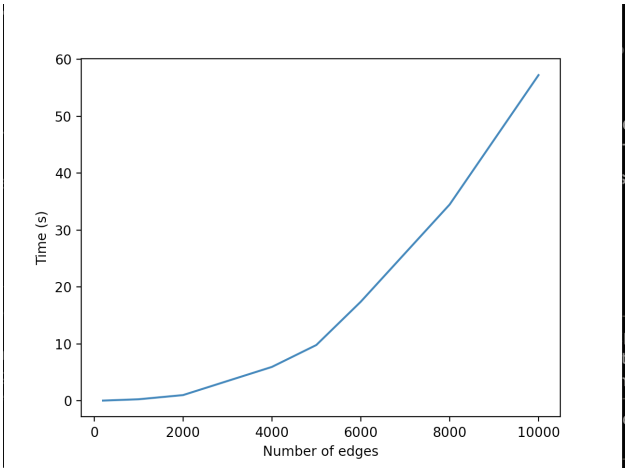
Probabilities using Power Iteration:

[[0.5 0.5]]

Running Time

We ran the PageRank Algorithm (generating the transition matrix and computing the eigenvector using Numpy methods and here are the results obtained against the number of nodes and edges.

Number of nodes	Number of edges	Time taken (s)
100	198	0.016000986099243164
500	998	0.24541902542114258
1000	1998	0.9779431819915771
2000	3998	5.930091142654419
2500	4998	9.780675888061523
3000	5998	17.391530990600586
4000	7998	34.47745680809021
5000	9998	57.222944021224976



As we can see, the time taken is approximately quadratic in terms of the number of edges in the graph.

PART B: Implementation of the HITS Algorithm

HITS algorithm was implemented on a directed networkx graph with web content, which contains 100 nodes and 256 directed edges.

The implementation used numpy and python to calculate the authority and hub scores by calculating the left eigenvector, i.e. the eigenvector corresponding to the largest eigenvalue.

We have used 2 methods to find the eigenvectors – using numpy.linalg.eig(), and using the power iteration method.

Functions Used

Generating the root and base sets:

```
generate_sets(query_word, postings_list)
```

Generates the root and base sets for the query word. First, it takes the entire posting list of the query word as root set, and initialises the base set with root set. Then iterates through all the edges in the given graph. If at least one of them is in the root set, it adds the other to the base set.

Generating the adjacency matrix:

```
generate_adj_matrix(base_set)
```

Generates the adjacency matrix. Takes the base set, then iterates through all the edges in the edge set. If both endpoints of some edge (a,b) are present in base set, it sets adj[a][b] to 1, indicating that there's an edge from a to b.

Generating hub and authority scores (using numpy.linalg):

```
generate_hub_authority_scores(adj_matrix)
```

Generates the hub and authority scores. It takes the adjacency matrix and then finds the left eigenvector directly using numpy.linalg.eig(), i.e. the eigenvector corresponding to the largest eigenvalue, of (AT)A to find the authority scores, and of A(AT) to find the hub scores. Here A denotes the adjacency matrix, and AT denotes its transpose.

Generating hub and authority scores (using power iteration):

`generate_hub_authority_scores_power_iteration(adj_matrix)`

Generates the hub and authority scores.

It takes the adjacency matrix and then finds the left eigenvector using the power iteration method, i.e. the eigenvector corresponding to the largest eigenvalue of (AT)A to find the authority scores, and of A(AT) to find the hub scores. Here A denotes the adjacency matrix, and AT denotes its transpose.

For this, we initialise our authority and hub scores with 1/n, where n is the number of nodes. Then, we multiply authority and hub scores by aTa and aaT respectively, and normalise the result by dividing it by the largest value in the matrix.

We do these iterations until the change after iteration becomes smaller than some threshold value. However, in this implementation we have run the iteration 1000 times, as this generally guarantees convergence.

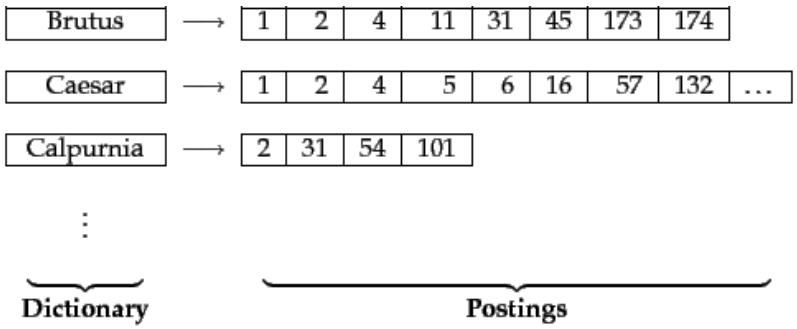
Then we divide the resulting scores with the sum of all scores, to bring them in the range [0,1], and then return them.

Data Structures Used:

Postings List:

Postings list is a list of all the terms inout dataset, and each word in the list has a corresponding list of documents containing that word.

Our implementation uses postings list to find the root set for a given query, as root set is defined as the set of nodes/pages that contain the given term.



Adjacency Matrix:

Adjacency is a way to represent a graph in a matrix format. It can be used to represent both weighted and unweighted graphs, as well as directed and undirected graphs.

We use an adjacency matrix of size = number of elements in base set, to represent the subgraph consisting only of nodes in the base set.

If Adj[i][j] = 1, it indicates that theres a directed edge from node i to j.

We use adjacency matrix in our code fo find the authority and hub scores as authority score is the left eigenvector of AdjT . Adj, and hub score is left eigenvector of Adj . AdjT.

Results

Testcase 1:

```
Query: credit
Top 3 authority scores are:
  Node    Authority Score
------  -----------------
    66           0.366025
     8           0.300098
    77           0.187538
Top 3 Hub scores are:
  Node    Hub Score
------  -----------
    15     0.384586
    77     0.366025
     8     0.1649
 List of all scores:
  Node    Authority Score    Hub Score
------  -----------------  -----------
     8           0.300098    0.1649
    66           0.366025    0.0844886
    77           0.187538    0.366025
    15           0.146339    0.384586
Authority score sum = 0.9999999999999999
Hub score sum = 1.0
```

Testcase 2:

```
Query: pension
Top 3 authority scores are:
  Node    Authority Score
------  -----------------
     0           0.320012
     3           0.21121
    61           0.21121
Top 3 Hub scores are:
  Node    Hub Score
------  -----------
    10     0.320012
    75     0.320012
     0     0.204815
 List of all scores:
  Node    Authority Score    Hub Score
------  -----------------  -----------
     0           0.320012     0.204815
     3           0.21121      0
    10          -0            0.320012
    75           0.128785     0.320012
    87           0.128785     0.155162
    61           0.21121      0
Authority score sum = 1.0
Hub score sum = 1.0
```

Running time

We ran the HIT algorithm for the various queries, and noted the base set size and time taken.

Query	Base Set Size	Time taken (s)
monday	41	0.011805295944213867
tuesday	56	0.012866020202636719
said	59	0.012999296188354492
new	69	0.012335777282714844
reuters	71	0.012788772583007812
business	74	0.013492822647094727
sports	75	0.024000167846679688
scitech	78	0.01341700553894043
world	82	0.022000789642333984