Assignment 6

Problem Statement:

Implement Page Rank Algorithm. (Use python or beautiful soup for implementation)

Objective:

- Evaluate and analyze retrieved information using Page Ranking algorithm.
- To study different Random Walk Methods.

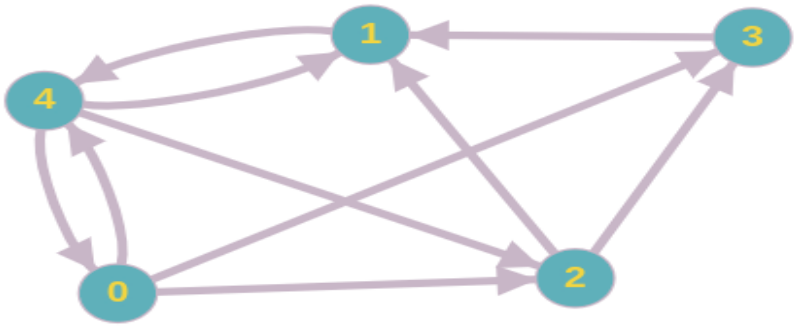
Theory:

Prerequisite: Page Rank Algorithm and Implementation, Random Walk.

In Social Networks page rank is a very important topic. Basically, page rank is nothing but how WebPages are ranked according to its importance and relevance of search. All search engines use page ranking. Google is the best example that uses page rank using the web graph.

Random Walk

The web can be represented like a directed graph where nodes represent the web pages and edges form links between them. Typically, if a node (web page) ***i*** is linked to a node ***j***, it means that ***i*** refers to ***j***.



Example of a directed graph

We have to define, the importance of a web page. As a first approach, we could say that it is the total number of web pages that refer to it. If we stop to these criteria, the importance of these web pages that refer to it is not taken into account. In other words, an important web page and a less important one have the same weight. Another approach is to assume that a web page spread its importance equally to all web pages it links to. By doing that, we can then define the score of a node ***j*** as follows:

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

where r_i is the score of the node ***i*** and d_i its out-degree.
From the example above, we can write this linear system:

$$\begin{cases} r_0 = \frac{r_4}{3} \\ r_1 = \frac{r_2}{2} + \frac{r_4}{3} + r_3 \\ r_2 = \frac{r_0}{3} + \frac{r_4}{3} \\ r_3 = \frac{r_2}{2} + \frac{r_0}{3} \\ r_4 = \frac{r_0}{3} + r_1 \end{cases}$$

By passing the right-hand side of this linear system into the left-hand side, we get a new linear system that we can solve by using Gaussian elimination. But this solution is limited for small graphs. Indeed, as this kind of graphs are sparse and Gauss elimination modifies the matrix when performing its operations, we lose the sparsity of the matrix and it would take more memory space. In the worst case, the matrix can no longer be stored.

Markov Chain and Page Rank:

Since a Markov Chain is defined by an initial distribution and a transition matrix, the above graph can be seen as a Markov chain with the following transition matrix:

$$P = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & \frac{1}{2} & 1 & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 1 & 0 & 0 & 0 \end{pmatrix}$$

Transition matrix corresponding to our example

We notice that ***P*** transpose is row stochastic which is a condition to apply Markov chain theorems. For the initial distribution, let's consider that it is equal to :

$$\pi = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}, \frac{1}{n})$$

where ***n*** is the total number of nodes. This means that the random walker will choose randomly the initial node from where it can reach all other nodes.
At every step, the random walker will jump to another node according to the transition matrix. the probability distribution is then computed for every step. This distribution tells us where the random walker is likely to be after a certain number of steps. The probability distribution is computed using the following equation:

$$\pi^{(t+1)} = P\pi^{(t)}$$

A stationary distribution of a Markov chain is a probability distribution π with $\pi = P\pi$. This means that the distribution will not change after one step. It is important to note that not all Markov chains admit a stationary distribution.
If a Markov chain is strongly connected, which means that any node can be reached from any other node, then it admits a stationary distribution. It is the case in our problem. So, after an infinitely long walk, we know that the probability distribution will converge to a stationary distribution π.
All we have to do is solving this equation:

$$\pi = P\pi$$

We notice that π is an eigenvector of the matrix P with the eigenvalue 1. Instead of computing all eigenvectors of P and select the one which corresponds to the eigenvalue 1, we use the ***Frobenius-Perron*** theorem.
According to ***Frobenius-Perron*** theorem, if a matrix A is a square and positive matrix (all its entries are positive), then it has a positive eigenvalue r, such as $|\lambda| < r$, where λ is an eigenvalue of A. The eigenvector v of A with eigenvalue r is positive and is the unique positive eigenvector.
In our case, the matrix P is positive and square. The stationary distribution π is necessarily positive because it is a probability distribution. We conclude that π is the dominant eigenvector of P with the dominant eigenvalue 1.
To compute π, we use the power method iteration which is an iterative method to compute the dominant eigenvector of a given matrix A. From an initial approximation of the dominant eigenvector b that can be initialized randomly, the algorithm will update it until convergence using the following algorithm:

Algorithm 1 Power method

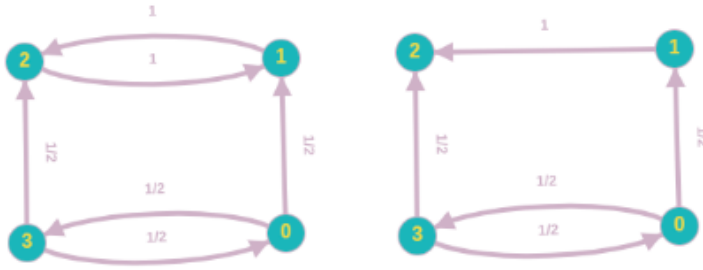
```
while (not converge) do
    b = A*b
    norm = compute_norm(b)
    b = b/norm
end while=0
```

Power method algorithm

As mentioned before, the probability distribution at time t defines the probability that the walker will be in a node after t steps. It means that the higher the probability, the more important is the node. We can then rank our web pages according to the stationary distribution we get using the power method.

Teleportation and damping factor

In the web graph, for example, we can find a web page i which refers only to web page j and j refers only to i. This is what we call ***spider trap problem***. We can also find a web page which has no outlink. It is commonly named ***Dead end***.



Dead ends and spider traps illustration

In the case of a spider trap, when the random walker reaches the node ***1*** in the above example, he can only jump to node ***2*** and from node ***2***, he can only reach node ***1***, and so on. The importance of all other nodes will be taken by nodes ***1*** and ***2***. In the above example, the probability distribution will converge to $\pi = (0, 0.5, 0.5, 0)$. This is not the desired result.
In the case of Dead ends, when the walker arrives at node ***2***, it can't reach any other node because it has no outlink. The algorithm cannot converge.
To get over these two problems, we introduce the notion of ***teleportation***.
Teleportation consists of connecting each node of the graph to all other nodes. The graph will be then complete. The idea is with a certain probability β, the random walker will jump to another node according to the transition matrix ***P*** and with a probability $(1-\beta)/n$, it will jump randomly to any node in the graph. We get then the new transition matrix ***R***:

$$R = \beta P + (1 - \beta))ve^T$$

where ***v*** is a vector of ones, and ***e*** a vector of ***1/n***

$$e^T = (\tfrac{1}{n}, \ldots, \tfrac{1}{n}).$$

$$v = (1, \ldots, 1)^T ;$$

β is commonly defined as the ***damping factor***. In practice, it is advised to set β to ***0.85***.
By applying teleportation in our example, we get the following new transition matrix:

$$\mathrm{R} = \begin{pmatrix} \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1}{3}\beta + \frac{1-\beta}{5} \\ \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1}{2}\beta + \frac{1-\beta}{5} & \beta + \frac{1-\beta}{5} & \frac{1}{3}\beta + \frac{1-\beta}{5} \\ \frac{1}{3}\beta + \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1}{3}\beta + \frac{1-\beta}{5} \\ \frac{1}{3}\beta + \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1}{2}\beta + \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1-\beta}{5} \\ \frac{1}{3}\beta + \frac{1-\beta}{5} & \beta + \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1-\beta}{5} & \frac{1-\beta}{5} \end{pmatrix}$$

Our new transition matrix

The matrix ***R*** has the same properties than ***P*** which means that it admits a stationary distribution, so we can use all the theorems we saw previously.

That's it for the PageRank algorithm. I hope you understood the intuition and the theory behind the PageRank algorithm. Please, do not hesitate to leave comments or share my work.

Random Walk Method – In the random walk method we will choose 1 node from the graph uniformly at random. After choosing the node we will look at its neighbors and choose a neighbor uniformly at random and continue these iterations until convergence is reached. After N iterations a point will come after which there will be no change In points of every node. This situation is called convergence.

Algorithm: Below are the steps for implementing the Random Walk method.

- Create a directed graph with N nodes.
- Now perform a random walk.
- Now get sorted nodes as per points during random walk.
- At last, compare it with the inbuilt PageRank method.
- Below is the python code for the implementation of the point's distribution algorithm.

Output:
PageRank using Random Walk Method.
[9 10 4 6 3 8 13 14 0 7 1 2 5 12 11]
PageRank using inbuilt pagerank method
9, 10, 6, 3, 4, 8, 13, 0, 14, 7, 1, 2, 5, 12, 11,

Oral Questions-
What is mean by Random Walk?
How you have implemented random walk method?
What is mean by Page Rank Algorithm?

Conclusion:
By this way, we have Implemented Page Rank Algorithm.