Numerical Optimization and Linear Algebra

Assignment 3

Mouselinos Spyridon – Part Time

1) Explanatory Analysis

In this exercise we are required to implement the PageRank algorithm and explore 2 different methods of calculating it:

- Via the Power Method.
- Via the formulation of an Iterative Solution with the use of Gauss Seidel Method.
-

2) On the Data Structures

In this task the data provided came into a file named "stanweb.dat" that contained 3 columns. The first column was the id of the web page that had an outgoing link to the id of the respective page on the second column, and the third column was a weight that showed the "importance" of that connection.

In fact, this data structure had to be reformed into a connectivity matrix of the graph of the webpages. Each webpage has its own id (node in the graph), while the connections to the other nodes (edges) have weight equal to 1/ (The number of outgoing connections from the starting node). In this formulation the sum of the outgoing edges of a node sum to 1 and represent the probability of the transition from the start node to the end node. As one can imagine this type of structure will contain a lot of zeroes in it. For this purpose, we decide to use the CSR Matrix Format, that is optimal in the way of storing Sparse Matrices for columnar operations. This is the type of Matrix-Vector operations that we will use in the rest of the exercise.

3) The PageRank Problem

In order to solve the problem of calculating the PageRank of each Node, the problem is formulated as a Markov Chain Transition Chain. However, if some rows contain all zeros, thus some nodes do not have any outbound links, the assumption that each row of the matrix represents a probability vector is false, making our matrix not stochastic. For this purpose, the hyperparameter ***alpha*** is used that gives a standard fraction of weight to each outbound connection regardless of the existence of a real webpage to webpage connection. This has to be done however not only on dangling (no outbound links) nodes but to every node in the graph, so the irreducibility of the Markov Chain is not lost. The new transition matrix is now:

$$\hat{P} = \alpha P + (1-\alpha)eeT/n$$

Where e is the uniform 1 vector. In some formulations $1/ne^T$ is replaced by v^T, named the personalization vector.

4) The Power Method

Our first method towards computing the PageRank vector is to view the problem as the eigenvector problem, $\pi^T \hat{P} = \pi^T$.

For any starting vector $x(0)^T$, we iteratively calculate:

$$x(k)^T = alpha\, x(k-1)^T P + \left(alpha\, x(k-1)^T a + (1 - alpha)\right) v^T$$

Where x is the eigenvector, k is the number of current iteration, a is a vector with 1 for nodes have only incoming connections and no outgoing connections, and v is the personalization vector mentioned above.

For speeding up our computations, we restate the problem as:

$$x(k) = (alpha\, x(k-1)^T P)^T + (alpha\, x(k-1)^T a v^T + (1 - alpha) v^T)^T$$

$$x(k) = alpha\, P^T x(k-1) + \alpha\left(alpha\, x(k-1)\right) v + (1 - alpha) v$$

In this way we do not need to calculate the Transpose of the intermediate results at each step and propagate it to the next. This is also the reason we opt for the Columnar Sparse Format, since we will make use of the P^T matrix and not P, while using the above computations.

5) The Gauss Seidel Method

In our second method we formulate the problem as a linear system. The eigenvalue problem can be rewritten as:

$$\pi^T (I - alpha\, P) = (a - alpha) v^T$$

This system is always accompanied by the normalization equation $\pi^T e = 1$, in order for the iterative solutions to maintain the constraint of representing probability vectors.

In this way we already know that linear systems can be solved by iterative methods and we opt to use the Gauss Seidel that uses:

$$M = I - (Low\ Trig\ of\ alpha\ *\ P^T)$$

$$R = (Upper\ Trig\ of\ alpha\ *\ P^T)$$

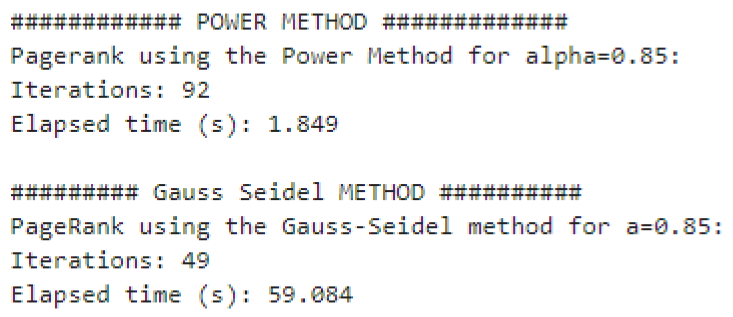
6) Termination Criteria

For both of the abovementioned methods we used the $l1$ norm of the difference between 2 consecutive pagerank vectors $x(k) – x(k-1)$. We use as a threshold the value $\tau = 1e-8$. The same threshold method is also implemented to keep track of the convergence iterations of each node in each method.

These however are the $l1$ norm of the difference between the respective index value of a node in the pagerank vector.

7) Question 1.A: Find vector π using $\alpha = 0.85$ and $\tau = 10^{-8}$

These are the results for the experiment:



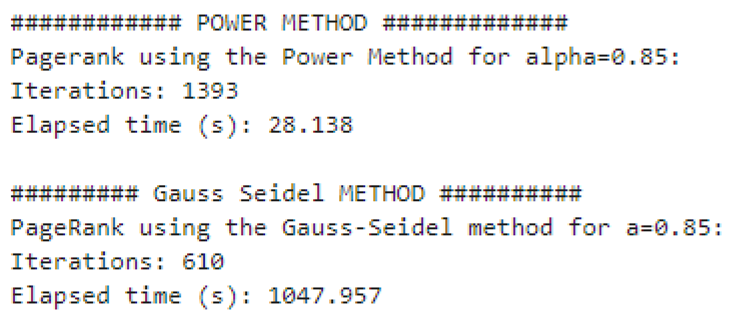
```
############ POWER METHOD #############
Pagerank using the Power Method for alpha=0.85:
Iterations: 92
Elapsed time (s): 1.849

######### Gauss Seidel METHOD ##########
PageRank using the Gauss-Seidel method for a=0.85:
Iterations: 49
Elapsed time (s): 59.084
```

We can observe that the power method converges faster, while the gauss seidel method uses fewer iterations to achieve the same convergence criteria. Also, by observing the Top PageRank Nodes after convergence we can see that the Top 10 , Top 25 nodes have the same index and values for both methods. However, at the Top 50 nodes the two methods start to differ in terms of the Node Ranking.

Rank	*PM Node Id*	*PM PageRank*	*GS Node Id*	*GS PageRank*
1	89073	0.01130	89073	0.01130
2	226411	0.00929	226411	0.00929
3	241454	0.00830	241454	0.00830
4	262860	0.00302	262860	0.00302
5	134832	0.00300	134832	0.00300
…	…	…	…	…
25	60210	0.00131	60210	0.00131
…	…	…	…	…
38	17781	0.00121	77999	0.00121

8) Question 1.B: Find vector π using $\alpha = 0.99$ and $\tau = 10^{-8}$

These are the results for the experiment:



```
############ POWER METHOD #############
Pagerank using the Power Method for alpha=0.85:
Iterations: 1393
Elapsed time (s): 28.138

######### Gauss Seidel METHOD ##########
PageRank using the Gauss-Seidel method for a=0.85:
Iterations: 610
Elapsed time (s): 1047.957
```

We can observe that here also the power method converges faster, while the gauss seidel method uses fewer iterations to achieve the same convergence criteria. Also, by observing the Top PageRank Nodes after convergence we can see that only the Top 10 nodes have the same index and values for both methods.

We can observe that the choice of the hyperparameter alpha at 0.99 brings extra burden to the convergence time of both methods. This is due to the fact that alpha simulates the importance of the linkage between the nodes, and as stated in the "Deeper Inside PageRank" paper, is the probability of a surfer following the link rather than simply teleporting to another webpage. The higher the alpha, the more austere is the system of equations we are faced to solve, and thus the more iterations and time any method utilized will require in order to oscillate around a convergence criterion.

This can be also seen in the Explanatory Analysis Between Alphas section of the exercise where one can observe that the change of alpha in the same method also causes another ranking in the nodes. Both the Power as well as the GS method have different ranking and pagerank values even in their respective Top 10 nodes with alpha= 0.85 and with alpha=0.99.

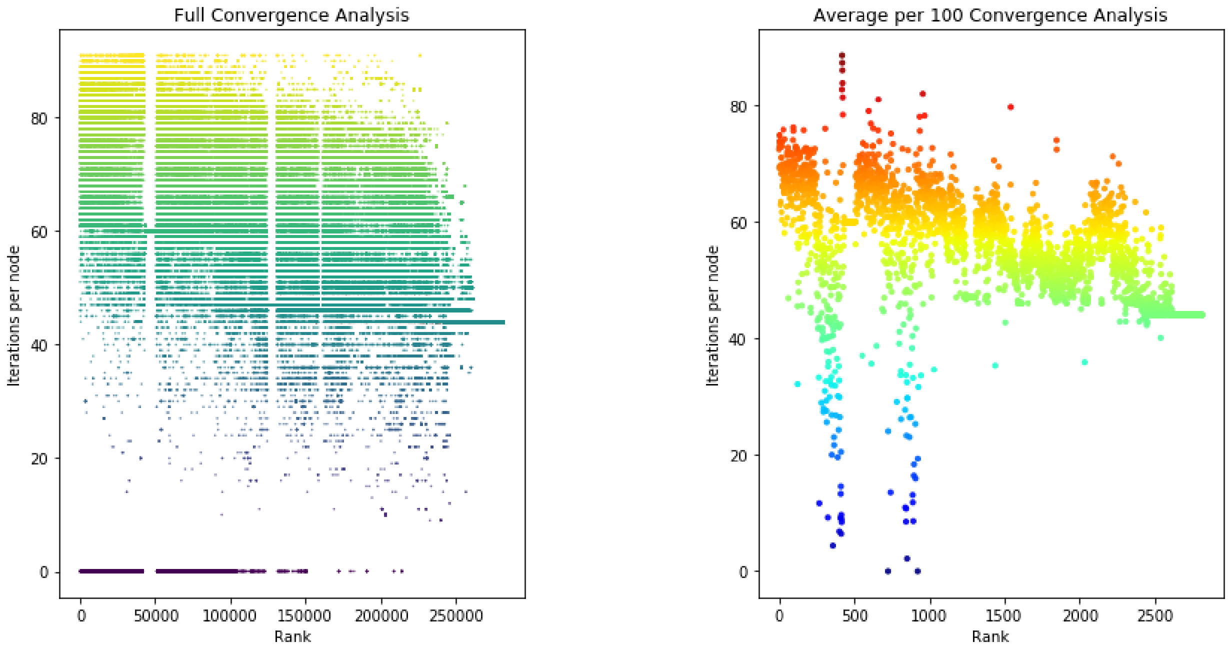
9) Question 1.C: Node Convergence Analysis

Below we present the plots of the iterations until convergence of each node vs the ranking of each node for all the abovementioned hyperparameter choices and methods.

The Left Image is the whole graph resulting plot while the Right Image is a per-100 node non-overlapping mean average of the same plot.

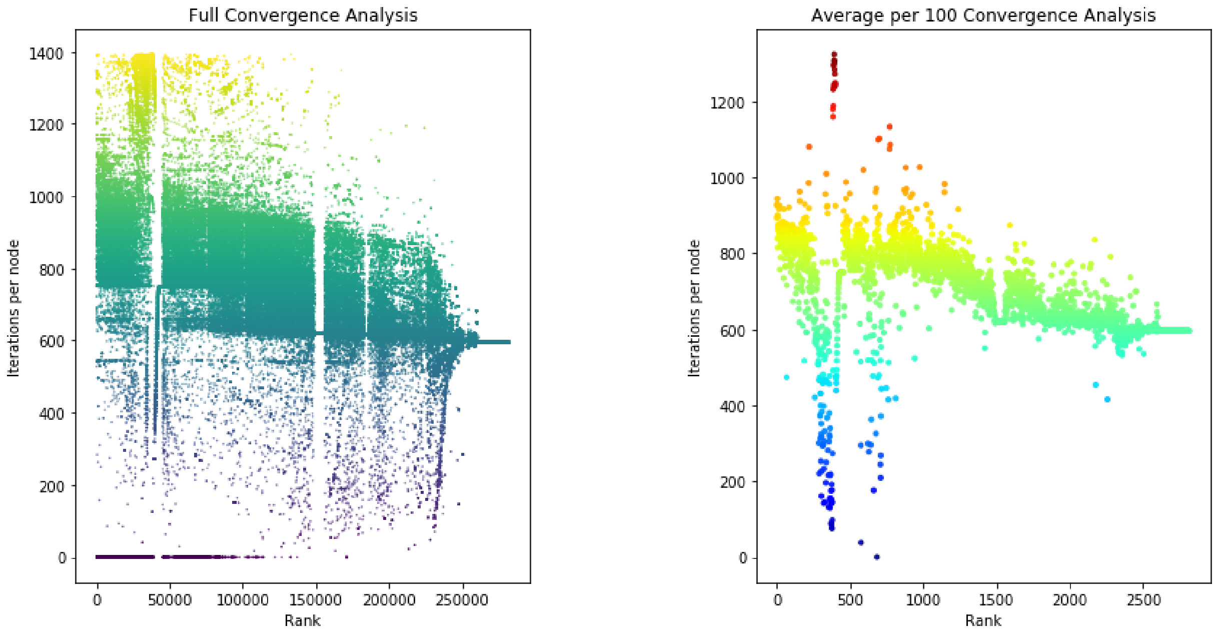
Power method for $\alpha = 0.85$

```
plot_convergence(pncv_pm_85,pi_vector_pm_85, name='pm_85')
```



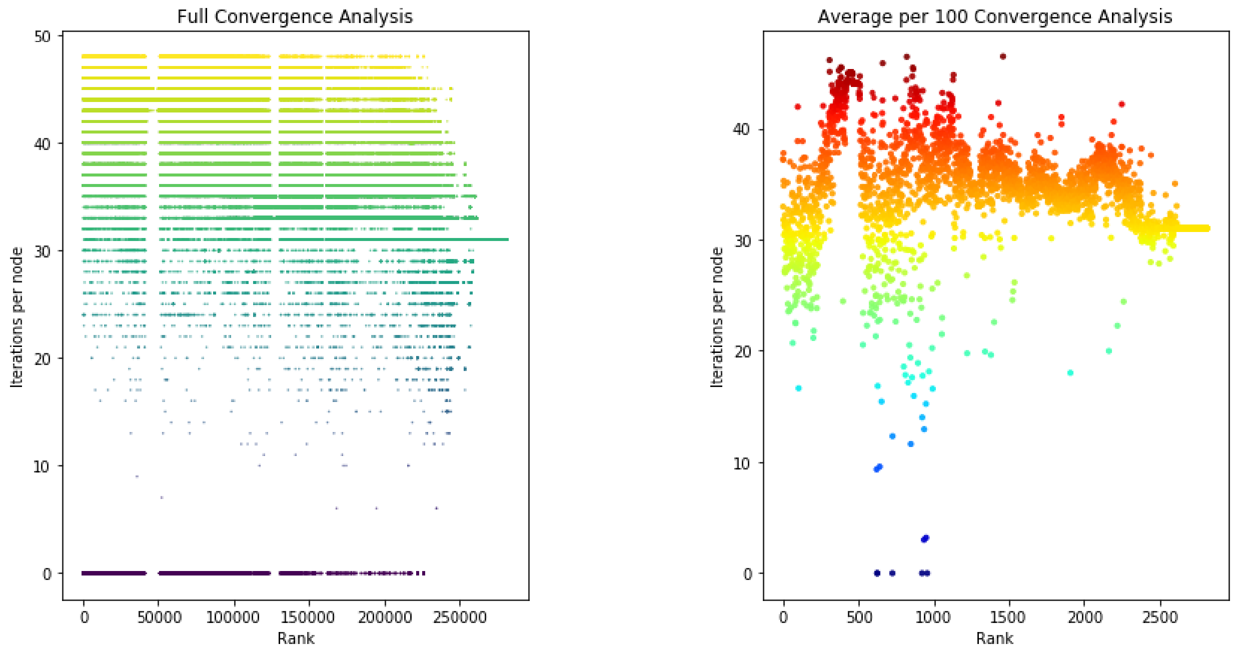
Power method for $\alpha = 0.99$

```
plot_convergence(pncv_pm_99,pi_vector_pm_99, name='pm_99')
```



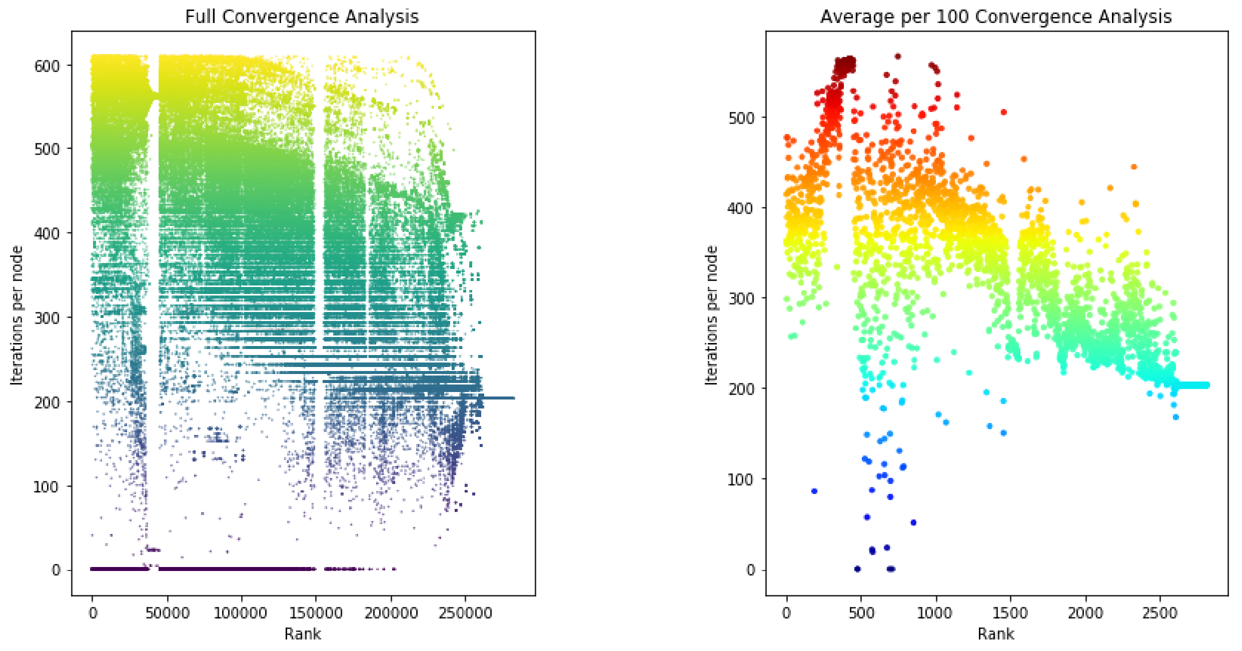
Gauss-Seidel method for $\alpha = 0.85$

```
plot_convergence(pncv_gs_85,pi_vector_gs_85, name='gs_85')
```



Gauss-Seidel method for $\alpha = 0.99$

```
plot_convergence(pncv_gs_99,pi_vector_gs_99, name='gs_99')
```



Both the above plots (full analysis and per 100 average) give us the following insight:

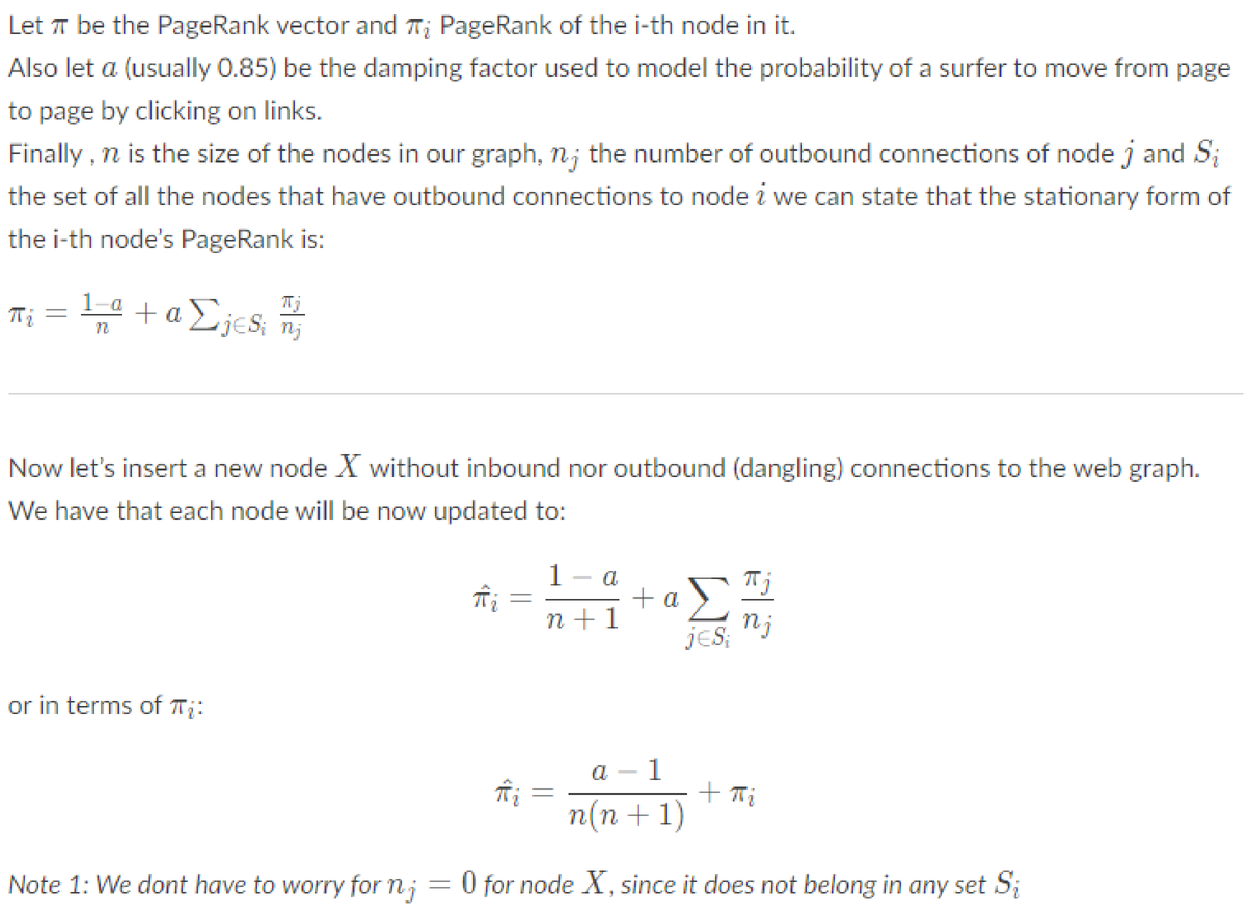
The Highest-Ranking Nodes need on average more iterations to converge than the Lowest-Ranking Nodes.

This is to no surprise, due to the fact that the low rank nodes tend to have more outgoing than incoming connections, thus "pushing" pagerank faster towards other nodes while their pagerank is converging fast to a value near the predefined threshold.

The opposite happens to the high rank nodes, where the vast incoming connections give boost to their pagerank constantly until a point where they fluctuate around the specific threshold.

10) Question 2.A)

The following questions are answered on the Jupyter Notebook, but also pasted here for completeness.



Let π be the PageRank vector and π_i PageRank of the i-th node in it.
Also let a (usually 0.85) be the damping factor used to model the probability of a surfer to move from page to page by clicking on links.
Finally , n is the size of the nodes in our graph, n_j the number of outbound connections of node j and S_i the set of all the nodes that have outbound connections to node i we can state that the stationary form of the i-th node's PageRank is:

$\pi_i = \frac{1-a}{n} + a\sum_{j \in S_i} \frac{\pi_j}{n_j}$

Now let's insert a new node X without inbound nor outbound (dangling) connections to the web graph.
We have that each node will be now updated to:

$$\hat{\pi}_i = \frac{1-a}{n+1} + a\sum_{j \in S_i} \frac{\pi_j}{n_j}$$

or in terms of π_i:

$$\hat{\pi}_i = \frac{a-1}{n(n+1)} + \pi_i$$

Note 1: We dont have to worry for $n_j = 0$ for node X, since it does not belong in any set S_i

Note 2: The upper applies vice-versa: Since X does not have any inbound nodes the respective set $S_i = \emptyset$

Thus, the PageRank of node X is :

$$\pi X = \frac{1-a}{n+1}$$

We can now make use of the fact that n is a very large number and for $n \to \infty$ we have:

$$\lim_{n\to\infty} \hat{\pi}_i = \lim_{n\to\infty} [\frac{a-1}{n(n+1)} + \pi_i] = \pi_i$$

We can so claim that the PageRanks of the older pages-nodes do not change.

11) Question 2.B)

We now add a node Y without inbound nodes *(so we can already calculate its PageRank just like above)* but with outbound nodes linking to X.

The PageRank of a random node i is updated to:

$$\tilde{\pi_i} = \frac{1-a}{n+2} + a\sum_{j\in S_i}\frac{\hat{\pi_j}}{n_j} \simeq \frac{1-a}{n+2} + a\sum_{j\in S_i}\frac{\pi_j}{n_j}$$

The PageRank of node X is now:

$$\pi\tilde{X} = \frac{1-a}{n+2} + a\pi\tilde{Y}$$

Note 3: The upper applies since only node Y belongs in X's inbound set

---

The PageRank of node Y is as expected:

$$\pi\tilde{Y} = \frac{1-a}{n+2}$$

And finally we can see that the PageRank of node X increases in comparison with the previous step at value:

$$\pi\tilde{X} - \pi\hat{X} = (a+1)\frac{1-a}{n+2} - \frac{1-a}{n+1} = (1-a)\frac{a(n+1)-1}{(n+2)(n+1)} > 0$$

For the rest nodes and for $n \to \infty$ we get yet again:

$$\lim_{n\to\infty}\tilde{\pi_i} = \lim_{n\to\infty}[\frac{1-a}{n+2} + a\sum_{j\in S_i}\frac{\pi_i}{n_j}] = \pi_i$$

We can so claim that the PageRanks of the older pages-nodes do not change once again.

12) Question 2.C)

In the previous questions we showed that:

- Adding a dummy no-inbound no-outbound node X, does not affect any other node other than giving an arbitrary pagerank to itself.
- The PageRank of node X increases by adding a node Y having outbound links to node X and no inbound links.
- Thus in order to further increase the PageRank of node X we utilize an additional node Z having outbound links to node X.

The best trick would be to link node Y with node X and node Z with node X and create backward links from X to Y and Z respectively. In this way we "retrofeed" some of the PageRank back to our dummy pages and boost the X page even more.

13) Question 2.D)

Given n is a very large number, PageRank of node X depends almost completely on its inbound flow of its links.
Adding links from node X to older popular nodes will not change the PageRank of node X.
Adding links from nodes Y and Z to older popular nodes does not change their PageRank as well neither affects the boosting of our system.

14) Question 2.E)

In order to answer we must just "scale up" the answer of question D.
In order to create a large link farm, the optimal way would be the following:

- Create a target page to be boosted, named T.
- Create m pages with outbound links to page T, and create links from T to each one of them. Let y be the final boosted PageRank of target page.

Now the proof:

The PageRank of each of the m pages is:

$$a * \frac{y}{m} + \frac{1-a}{n}$$

The first term represents the contribution from t. The PageRank y of t is taxed, so only ay is distributed to t's successors. That PageRank is divided equally among the m supporting pages. The second term is the supporting page's share of the fraction 1 − a of the PageRank that is divided equally among all n pages.

Now let's compute y:

We have a times the PageRank of each Page above:

$$a(a\frac{y}{m} + \frac{1-a}{n})$$

plus:

$$(1-a)/n$$

the share of the fraction $1-a$ of the PageRank that belongs to
t. Since n is a relatively large number this quantity can be safely dropped.

Thus:

$$y = am(\frac{ay}{m} + \frac{1-a}{n}) = a^2y + a(1-a)\frac{m}{n}$$

Solving for y we get:

$$y = \frac{a}{1+a}\frac{m}{n}$$