PageRank Algorithm

Suman, Sucharita, Sourendu, Srijan

September 13, 2021

Abstract

Ranking the documents of a search engine is an important task and one which determines the popularity of an engine. In this report we describe an algorithm which is used by the Google Search engine. We compare this algorithm to an ideal web surfer and mathematically model the behavior which gives us the provable working pagerank algorithm. We further demonstrate an implementation of the same in Python.

Contents

1 Introduction

In this section you may answer questions such as:

- What is this report about?
- What is the context of this work?
- What work was done in this context earlier? (background and a short literature review)
- What is the work you have done (quick mention of implementations or demonstrating examples, observations etc.)?

In today's world, World Wide Web (Web) is the largest information repository consisting of set of nodes interconnected by hypertext links. The widespread use of Web is making it difficult to manage all the information. Nowadays, to search for a given topic, we open our favorite search engine, like Google, AltaVista, Yahoo, and type in the key words, and the search engine displays the pages relevant to the query. While for any given topic there might be thousands or even millions of pages available, the problem of ranking those pages to generate a short list is probably one of the key problems of Web Information Retrieval, and this requires some kind of relevant estimation.

At a glance, it appears that a search engine maintains an index of all web pages, and when a user types in a query, the engine browses through its index and counts the occurrences of the key words in each web file. The pages with the highest number of occurrences of the key words receives the highest score based on the traditional text retrieval model and gets displayed to the user. This used to be the picture in the '90s which only looked at the content of a document, but ignored its influence.

For example, if you search the word "Harvard" in your browser, you would expect that your search engine ranks the homepage of the Harvard University as the most relevant page. But if we create a web page that contains "Harvard" a million times, we can see that the word "Harvard" appears much more often into this website rather than the homepage of Harvard university. Therefore, the question is should we consider this page relevant to our query "Harvard"? The answer is of course not. Here PageRank algorithm comes to our rescue.

1.1 Motivation

One of the most influential algorithms for computing the relevance of web pages known as PageRank algorithm essentially solved the web search problem and was the basis for Google's success. It was introduced by Google founders Larry Page and Sergey Brin, two graduate students at Stanford University in 1998. It works on the principle "the more links, the more important the website".

2 Page Rank algorithm

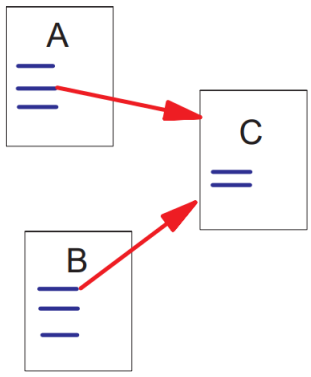
2.1 Definition

Page Rank is an algorithm used to measure the importance of website pages using hyperlinks between them. The aim of this algorithm is to track down the difficulties with the content based ranking algorithm of early search engines which used text documents for webpages to retrieve the information with no explicit relationship of link between them. It is used to give each page a relative score of importance and authority by evaluating the quality and quantity of its hyperlinks. Therefore, it can be said that Page rank is a "vote", given by all the other pages on the web to determine how relevant a page is. A link to a page counts as a "vote" of support.

2.2 Link Structure of the Web

Current estimates show that there are over 1.7 billion webpages all over the world. The Web's hyperlink structure forms a massive directed graph. The nodes in the graph represent web pages and the directed arcs or links represent the hyperlinks. Every page has some number of forward links(outedges) and back-links(inedges). So there are two kinds of hyperlinks inbound links and outbound links.

- **Inbound Links:** Inbound links are those links that come from other site to your website, also known as "backlinks". Google consider only those relevant links that point to one's site but one cannot control the number of sites that can point to its site. If the website content is unique and rich then there are much chances those links will be "dofollow" otherwise links will be considered as "nofollow".
- **Outbound Links:** Outbound links are those links that is pointing to other sites from one's website and one has more control over these links. A page has high rank if the other pages with high rank linked to it.



Here A and B are C's backlinks and C is A and B's forward link.

2.3 Propogation of Ranks through Webpages:

Initially, one link from a site is equaled to one vote for the site that it was linked to. Before one starts collecting links from several different pages, one should know the value of every link. The value of inbound links is determined by the number of outbound links from the linking site and its PageRank. That is, because the PageRank of the linking site is divided by the total number of outbound links the page has. A page that is linked to several relatively important websites (those with high PageRank themselves) has value, while a website that has no incoming links is considered unimportant. The more the incoming links from high-value websites you get, the better it is for the website. Intuitively, we might think that a web page is important if many other pages link to it; each link provides a "recommendation" that the page is worth visiting. However, not all links are created equal. A link from an important web page is more significant than a link from an unimportant web page. While sometimes this simple page rank calculation rule may be violated. For example, if a webpage has a link from the Yahoo homepage it may be just one link but it is a very important one. So, this webpage should be ranked higher among many other webpages which may have more links coming from less important and rare webpages.

Thus, from the above discussion, we come to the conclusion that a particular page has high rank if sum of ranks of all its backlinks is high. So there may be two cases over here:

- The first is the simple citation where a page may have many backlinks.
- The second case is the one where a page has few links but they are very highly ranked backlinks.

2.4 Calculation of PageRanks:

In order to give a simple page ranking R, a slightly simplified version of PageRank is given by:

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

- u be defined as a webpage.
- F_u be the set of webpages that u points to
- B_u be the set of webpages that points to u i.e. set of backlinks of u
- $N_v = |F_v|$ be the number of links from v i.e. the number of forward links from v
- c be the normalization factor.

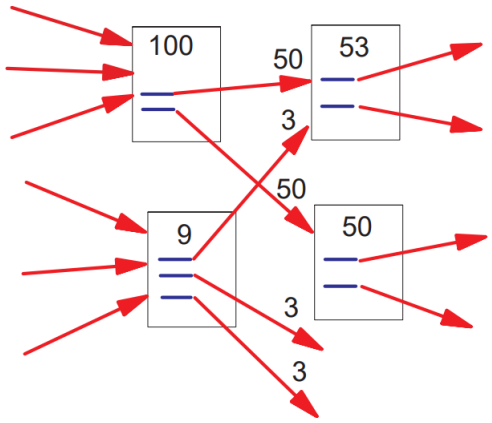


Figure 2: Simplified PageRank Calculation

In order to prevent some pages from having too much influence, the PageRank formula also uses a dampening factor also known as the normaliztion factor. The value of c is kept between 0 and 1. There maybe some number of pages with no forward links. To ensure that their weight is not lost from the system here c is taken less than 1.

Therefore, c can be defined as the probability by which a webpage follows the hyperlink structure. The equation is recursive and it can be computed by starting with any set of ranks and we iterate the computation until the rank converges. This is further explained with an example later.

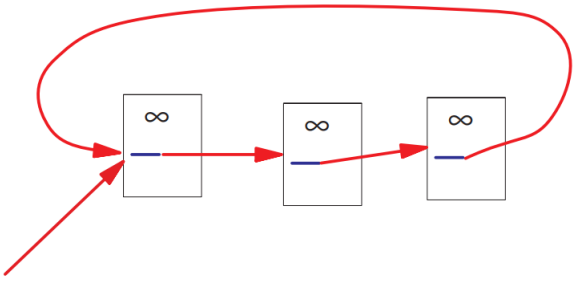


Figure 4: Loop Which Acts as a Rank Sink

But, there is a problem with this simplified ranking function. Suppose we have two webpages that point only to each other and not to any other page but some other pages may point to them. Then during each iteration, the loop formed by the two webpages accumulates rank but it is never able to distribute any rank to the other pages. The loop thus forms a sort of trap called rank sink i.e. if a page has no outgoing links to other pages, it is a sink. In order to overcome this, we introduce a rank source. Our goal is to distribute the rank of sink pages among all pages in the graph.

2.4.1 Modified Version of PageRank

Let $E(u)$ be some vector over the webpages that corresponds to a source of rank. So, now the PageRank of a new webpage can be assigned a ranking R' satisfying the below equation.

$$R'(u) = c_1 \sum_{v \in B_u} \frac{R'(v)}{N_v} + c_2 E(u)$$

- $c_2 = (1 - c_1)$
- $E(u)$: A distribution of ranks of web pages that users jump to when they get bored after successive links at random.

So, c_1 is the probability by which pagerank of a webpage propagates to its neighbours and c_2 is the probability of not following the provided data node links and just randomly pointing to some other node. There are two ways to overcome the rank sink:

- We can add outgoing edges from sinks to all pages.
- We can distribute a sink's pagerank evenly among all pages.

Functionally, the two methods should yield identical results. After we have determined which pages are sinks, we can add outgoing edges from each sink to every page in our graph, including the sinks.

2.5 Random Walks in a Graph

We now discuss an alternative approach one could follow for ranking pages on the Web, which is known as the Random Surfer Model. This will turn out to be basically equivalent to the algorithm we discussed before. One can imagine a user browsing the Web. The user starts at an arbitrary page, and follows links at random, until he gets stuck (i.e. comes to a page with no outgoing links). This can be modelled as a random walk on the web graph. At each step, the user picks an outgoing edge at random. The importance of the page is determined by the proportion of time spent by the user on that page.

2.5.1 Random Surfer Model

- A random surfer is one that bounces along randomly following the hyperlink structure of the Web.
- It corresponds to the standing probability distribution of a random walk on the graph of the web.
- The random surfer simply keeps clicking on successive links at random forever or until it gets stuck.
- The random surfer chooses one of the several outlinks present in a page.

2.5.2 Modified Random Surfer Model

However, if a real web surfer ever gets into a small loop of web pages, the surfer will not continue in the loop forever. Nowadays, the web surfer gets worn out too early on the web because of non-availability of relevant information and they can easily teleport to new web pages rather than following hyperlink structure. In the modified model, the random surfer simply keeps clicking successive links at random, but periodically gets bored and jumps to a random page based on the distribution of E. To model the user behaviour E is taken uniform over all the webpages.

3 Main results

In this section you mention the main results of the paper you have studied, with whatever prerequisites (definitions, lemmas etc) are needed. You should write an outline of the proof, or the mention the main ideas that the proof is based on here. You are encouraged to write your version of the proof (if you can/wish to).You are also encouraged to simplify the exposition of the paper in your own words here.

3.1 A Mathematical Model

We now focus on how the page rank algorithm would work and why it would give a vector of scores.
As seen above the random surfer model can be viewed as a Markov chain. We present a definition of Markov chains below:
Let $\mathbb{X} = \{X_i \mid i \in \mathbb{N}\}$ be a sequence of random variables which take values from a countable set S (Usually $S \subset \mathbb{R}^d$ or $S \subset \mathbb{Z}^d$ for some $d \in \mathbb{Z}^+$). Then $\mathbb{X}$ is said to be a Markov Chain with State Space S if $\forall n \in \mathbb{N},\ \forall\ j_0, \ldots, j_{n+1} \in S$,

$$Pr(X_{n+1} = j_{n+1} \mid X_n = j_n, \ldots, X_0 = j_0) = Pr(X_{n+1} = j_{n+1} \mid X_n = j_n)$$

A bit more terminology: If $\mathbb{X}$ is indeed a Markov Chain with State Space S then the elements of S are called states.

3.2 Time matters

We can easily see that, given an initial site at time 0, the probability a random surfer would move to another particular site at time t would only depend on the site the surfer was in at time $t-1$ because the surfer would need to click on one of the outgoing links itself. So, $P(X_t = i \mid X_{t-1} = i_{t-1}, \ldots, X_0 = i_0)$ is just $P(X_t = i \mid X_{t-1} = i_{t-1})$ which is what a Markov chain is.
Observe, that based on the random surfer model, the probability a random surfer would move to another particular site at time t would only depend on the site the surfer was in at time $t-1$ and further more it is independent on the value of t. Basically what we are saying is that our Markov chain model is time homogeneous.

A Markov Chain is said to be time homogeneous if $\forall n \in \mathbb{N},\ \forall i, j \in S$

$$Pr(X_{n+1} = i \mid X_n = j) = Pr(X_1 = i \mid X_0 = j)$$

Thus, we have our random surfer as a time homogeneous Markov chain. We now look at a few other definitions that will aid us later.

We state a few definitions on matrices below.
Let $P = ((p_{ij}))_{i,j \in S}$ be a square matrix. Then, P is said to be a row-stocastic matrix whenever the following 2 conditions hold

1. ("Elements are positive") $\forall i, j \in S$, $(p_{ij}) > 0$
2. ("Rows sum to 1") $\forall i \in S$, $(\sum_{j \in S} p_{ij}) = 1$

In a similar vein, Let $P = ((p_{ij}))_{i,j \in S}$ be a square matrix. Then, P is said to be a column-stocastic matrix whenever the following 2 conditions hold

1. ("Elements are positive") $\forall i, j \in S$, $(p_{ij}) > 0$
2. ("Columns sum to 1") $\forall j \in S$, $(\sum_{i \in S} p_{ij}) = 1$

Consider a time homogeneous Markov Chain $\mathbb{X} = \{X_i \mid i \in \mathbb{N}\}$ on a state space S.
Let $p_{ij} := Pr(X_1 = i \mid X_0 = j)$ where $i, j \in S$.
Then, $P = ((p_{ij}))_{i,j \in S}$ is said to be the column-stocastic matrix of $\mathbb{X}$

For websites i, j we define P_{ij} as the probability $P(X_1 = i \mid X_0 = j)$ and so P stores all the probabilities of moving from one website to another. We can call this P as a hyperlink matrix. We can also see that this P is the column-stocastic matrix of the random surfer model - modelled as a Markov Chain.

3.3 Calculating the paths

To find the probability of reaching a website i from website j in any number of paths in a graph of n nodes (here, websites) takes $O(n^2)$ and doing that for all websites would then take $O(n^4)$. Matrix computations have come far, calculating the square of a $n \times n$ matrix takes just $O(n^{2.4})$ time. So overall computing powers of matrices would be a lot better. So, we now present a definition which will aid us later.

Let P be the column-stocastic matrix of time-homogeneous Markov Chain $\{X_i \mid i \in \mathbb{N}\}$.
$\forall n \in \mathbb{N}$, we define $P^{(n)}$ as $((p_{ij}^{(n)}))_{i,j \in S}$ where $\forall i, j \in S$ we define $((p_{ij}^{(n)}))$ as $Pr(X_n = i \mid X_0 = j)$

Now, for our hyperlink matrix P, if we can show that $P^{(n)} = P^n$ then we would be able to capture the probability a random surfer would reach some website i from some other website j in any number of paths from the powers $P, P^2, P^3, \ldots, P^n$ of P. The lemma below outlines a proof for exactly that.
Lemma. $\forall n \in \mathbb{N}$ $P^{(n)} = P^n$
Proof. We show it by induction on n. The base case $n = 1$ is already clear. Suppose this holds for some k. Then, $((p^{(k+1)}))_{i,j} = Pr(X_{k+1} = i | X_0 = j) = \sum_{s \in S} Pr(X_{k+1} = i | X_1 = s, X_0 = j) Pr(X_1 = s | X_0 = j)$.
Now, $Pr(X_{k+1} = i | X_1 = s, X_0 = j) = Pr(X_k = i | X_0 = s)$ and so we get:

$$Pr(X_{k+1} = i | X_0 = j) = \sum_{s \in S} Pr(X_k = i | X_0 = s) Pr(X_1 = s | X_0 = j)$$

therefore we get

$$Pr(X_{k+1} = i | X_0 = j) = \sum_{s \in S} ((p^k))_{is} ((p))_{sj}$$

which equals $(P^{(k)}P)_{ij} = (P^k P)_{ij} = (P^{k+1})_{ij}$.
Thus, $(P^{(k+1)})_{ij} = ((p^{(k+1)}))_{ij} = (P^{k+1})_{ij}$
Thus, by induction our lemma is proven.

This lemma basically shows that the hyperlink matrix we constructed and its powers is enough for taking account of all the probabilities of visiting various different sites from a particular website.

3.4 A bit more properties

Period of a state: The period $d(k)$ of a state k of a homogeneous Markov Chain with Transition matrix P is given by

$$d(k) = gcd\{m \geq 1 : P^m_{k,k} \geq 0\}$$

If $d(k) = 1$ then we call state k **aperiodic**
Aperiodic Markov Chain: A Markov Chain is called **aperiodic** if all its states are **aperiodic**
Aperiodic Markov Chains have a useful result:

Result: For an Aperodic finite Markov chain with transition matrix P $\exists$ a positive integer N s.t. $(P^m)_{i,i} > 0 \ \ \forall i$ and $\forall m \geq N$

Which means there will always be a minimum no. of steps such that state i could be reached from itself

Proof: To prove this result we need the help of a lemma based on Number Theory
Lemma: *If a subset A of natural no.s is s.t. A is closed under addition and* $gcd\{a|a \in A\} = 1$
then A contains all but finitely many non-negative integers so that $\exists n$ *s.t.* $\{n, n+1, n+2, \ldots\} \subseteq A$
Now, let A_i be the set of possible return times $\forall i \in S$
the Markov chain is aperiodic $\implies gcd(A_i) = 1$
If $m, m' \in A_i$ then,
$Pr[X_m = i|X_0 = i] > 0$ and $Pr[X_{m+m'} = i|X_m = i] > 0$
$\therefore Pr[X_{m+m'} = i|X_0 = i] \geq Pr[X_m = i|X_0 = i]Pr[X_{m+m'} = i|X_m = i] > 0$
$\implies m + m' \in A_i \implies A_i$ is closed under addition
$\therefore$ by the lemma A_i contains all but finitely many natural no.s
$\therefore A$ contains all but finitely many natural no.s
Hence proved
Communication Class: Two states i and j are said to **communicate with each other** iff $P^m_{i,j} > 0$ and $P^n_{j,i} > 0$ for some finite m, n
It can be easily observed that the communication relation is an equivalent relation. A Communication Class is a one that contains all the states that communicate with every other state of that class
Irreducible Markov Chain: A Markov chain is called **Irreducible** when all its states belong to a single communication class.
Which is to say that A Markov Chain is called Irreducible if all of its states are eventually accessible from all of its states after some finite no. of steps.

For a Markov Chain that is both Irreducible and Aperiodic we have the following result
Result: For an Irreducible Aperiodic finite state Markov Chain with transition matrix P we have the following result:

$\exists M < \infty$ s.t. $(P^m)_{i,j} > 0 \forall i, j$ and $\forall m \geq M$
Which means for an Irreducible Aperiodic finite state Markov Chain each state can be eventually reached from each other state in a finite no. of steps
Proof: As the Markov chain is Aperiodic, $\exists$ a positive integer N s.t. $(P^n)_{i,i} > 0 \ \ \forall i$ and $\forall n \geq N$
As the Markov chain is Irreducible, there exists a positive integer $n_{i,j}$ s.t. $P_{i,j}^{n_{i,j}} > 0$
After $m \geq N + n_{i,j}$ steps, we will have
$Pr[X_m = i | X_0 = j] \geq Pr[X_m = i | X_{m-n_{i,j}} = j] Pr[X_{m-n_{i.j}} = j | X_0 = j]$
or, $P_{i,j}^m \geq P_{i,j}^{n_{i,j}} \cdot P_{j,j}^{m-n_{i,j}}$
i.e we have $P_{i,j}^m > 0$, as we claimed
Hence proved

Steady-state Distribution: For a finite Markov Chain with Transition matrix P a column vector $w = (p_0, p_1, \ldots, p_{(n-1)})'$ will be a Steady-state distribution for P if and only if

- P is column stochastic
  and
- $Pw = w$
  such a vector must be an eigenvector of P corresponding to the eigenvalue 1

Proposition:
For an Irreducible Aperiodic finite state Markov Chain $\exists$ precisely one Steady-state Distribution

This proposition is true due the following theorem given by **Perron** and **Frobenius**
Further, it guarantees us starting from any initial probabilty vector q we will reach that Stedy-state distribution in a finite no. of steps

Perron Frobenius Theorem: **For an Irreducible Aperiodic Finite state Markov chain with transition matrix P the unique eigenvector corresponding to the eigenvalue 1 can be chosen as the probability vector w that satisfies $\lim_{t \to +\infty} P^{(t)} = [w, w, \ldots, w]$.**
Moreover, for any probability vector q we will have $\lim_{t \to +\infty} P^{(t)} q = w$

We have not provided any extensive proof of this theorem.It is usually proved by Spectral Theory considering the spectral radius of P matrix.

Idea for Implementation:
Let us have,
B_i be the set containing all the pages linking to page P_i ,
l_j be the no. of links from page P_j

Then the **Hyperlink Matrix** $H = ((H_{ij}))$ is definied as:

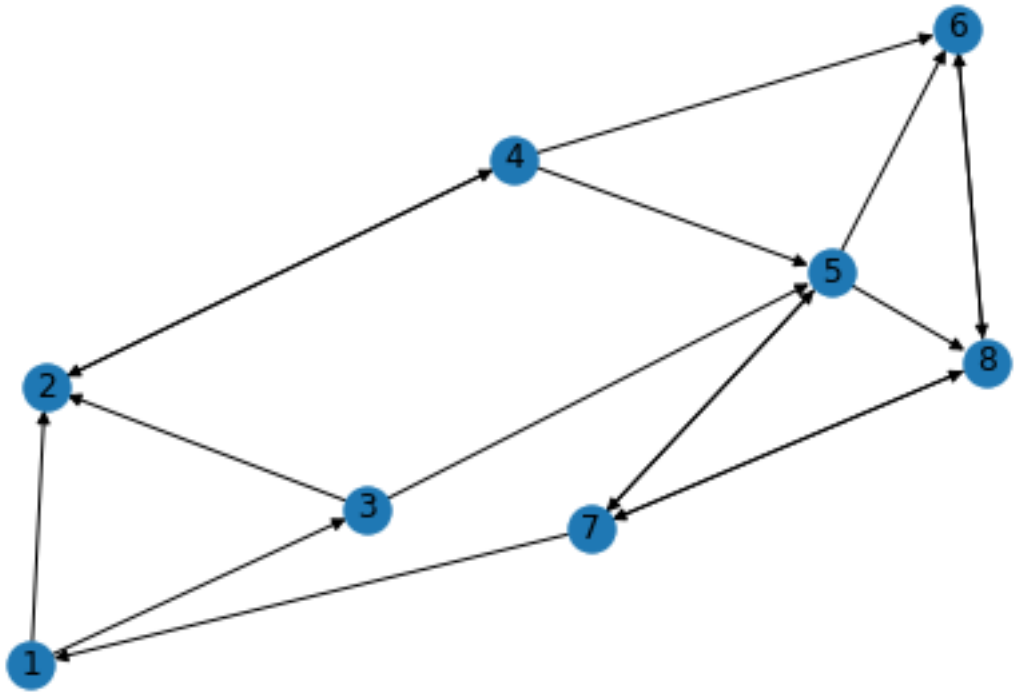
$$H_{ij} = 1/l_j \quad if \quad P_j \in B_i$$
$$= 0 \quad otherwise$$

This Hyperlink Matrix is usually neither Irreducible nor Aperiodic but the Google Matrix G is
We will later see how the Google Matrix is obtained from the Hyperlink Matrix to apply the results we have seen so far

4 Example

4.1 Simple Pagerank Algorithm

Let us take this directed graph for calculating Pagerank. The nodes of the graph are the webpages and the directed edges are forward link or backward link from one page to another.



The corresponding Hyperlink Matrix would be:

$$H = \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.33 & 0.00 \\ 0.50 & 0.00 & 0.50 & 0.33 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.50 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.50 & 0.33 & 0.00 & 0.00 & 0.33 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.33 & 0.33 & 0.00 & 0.00 & 0.50 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.33 & 0.00 & 0.00 & 0.50 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.33 & 1.00 & 0.33 & 0.00 \end{pmatrix}$$

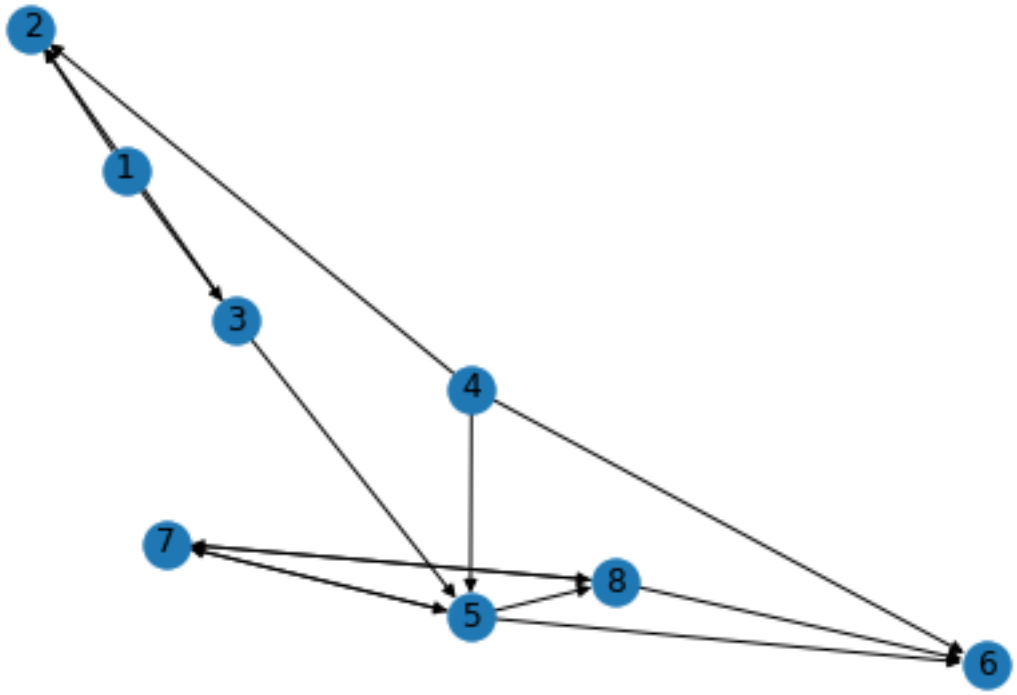
We have considered power method for calculating the eigenvector. In this method, we begin by choosing a vector $I^{(0)}$ (which is generally considered to be $(1, 0, 0, ..., 0)^{'}$) as a candidate for I and then produce a sequence of vectors $I^{(k)}$ such that

$$I^{(k+1)} = HI^{(k)}$$

In our case after 100 iterations the rank matrix would be $I = \begin{pmatrix} 0.060 \\ 0.067 \\ 0.030 \\ 0.068 \\ 0.098 \\ 0.202 \\ 0.180 \\ 0.295 \end{pmatrix}$

4.2 Dangling Nodes

Now we would try to demonstrate what are dangling nodes and what are the problems related to dangling nodes in Pagerank calculation.



The corresponding Hyperlink Matrix would be:

$$H = \begin{pmatrix} 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.500 & 0.000 & 0.500 & 0.333 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.500 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.500 & 0.333 & 0.000 & 0.000 & 0.500 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.333 & 0.333 & 0.000 & 0.000 & 0.500 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.333 & 0.000 & 0.000 & 0.500 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.333 & 0.000 & 0.500 & 0.000 \end{pmatrix}$$

In this case after 100 iterations the rank matrix would be $I = \begin{pmatrix} 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \\ 0.000 \end{pmatrix}$

- So, measure of importance of all the pages are zero. There is some problem!!
- If it is noticed properly the page 2 and 6 has no link going out.
- These takes some importance in each step and doesn't pass it to other pages. So gradually all the pages' measures of importance becomes 0.
- In this example Node 2 and 6 are Dangling Nodes.
- Pages with no links are called dangling nodes, and there are, of course, many of them in the real web.
- Now we have to modify H.
- If we surf randomly, then at some point we might end up at a dangling node.
- The hyperlink matrix H is modified by replacing the column of zeroes (if any) with a column in which each entry is $1/n$ where n is the total number of web pages.
- Let's denote the updated Hyperlink Matrix as S for future references.

The updated Hyperlink Matrix would be:

$$S = \begin{pmatrix} 0.000 & 0.125 & 0.000 & 0.000 & 0.000 & 0.125 & 0.000 & 0.000 \\ 0.500 & 0.125 & 0.500 & 0.333 & 0.000 & 0.125 & 0.000 & 0.000 \\ 0.500 & 0.125 & 0.000 & 0.000 & 0.000 & 0.125 & 0.000 & 0.000 \\ 0.000 & 0.125 & 0.000 & 0.000 & 0.000 & 0.125 & 0.000 & 0.000 \\ 0.000 & 0.125 & 0.500 & 0.333 & 0.000 & 0.125 & 0.500 & 0.000 \\ 0.000 & 0.125 & 0.000 & 0.333 & 0.333 & 0.125 & 0.000 & 0.500 \\ 0.000 & 0.125 & 0.000 & 0.000 & 0.333 & 0.125 & 0.000 & 0.500 \\ 0.000 & 0.125 & 0.000 & 0.000 & 0.333 & 0.125 & 0.500 & 0.000 \end{pmatrix}$$

Notice that previously all the elements of Column 2 and Column 6 were 0. Now those have been replaced by 0.125

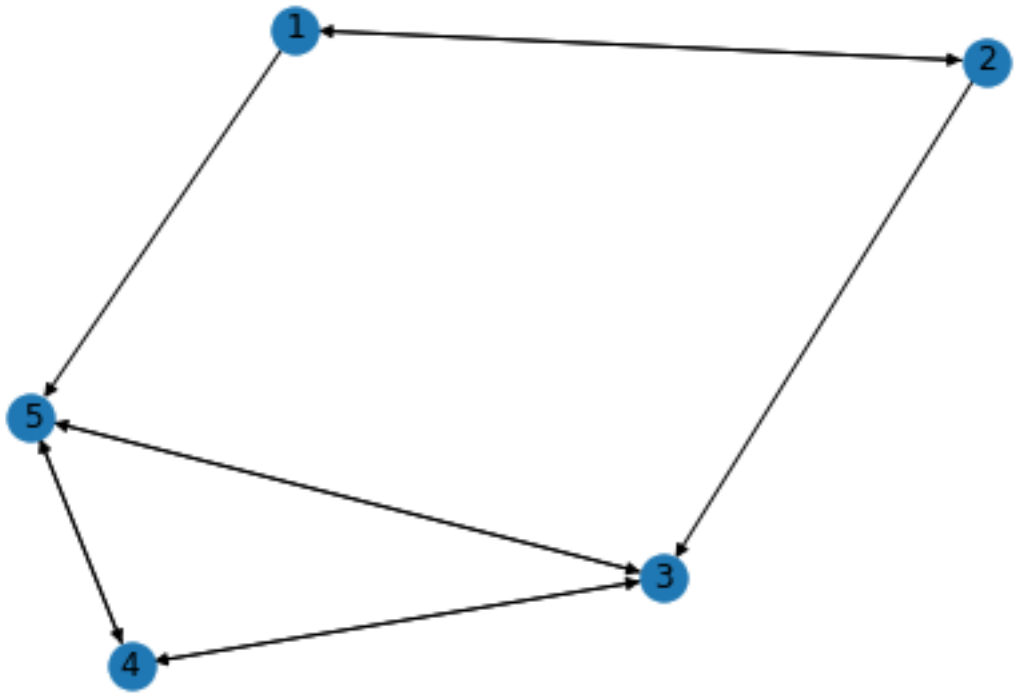
After updating the hyperlink matrix the new Pagerank matrix would be

$$I = \begin{pmatrix} 0.038 \\ 0.098 \\ 0.057 \\ 0.038 \\ 0.176 \\ 0.206 \\ 0.193 \\ 0.193 \end{pmatrix}$$

So the problem of Dangling Nodes solved!!

4.3 Rank Sink

Let us discuss about another problem which can occur frequently while we are calculating Pagerank. We have to calculate the page rank of the following graph:



The corresponding updated Hyperlink Matrix would be:

$$S = \begin{pmatrix} 0.000 & 0.500 & 0.000 & 0.000 & 0.000 \\ 0.500 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.500 & 0.000 & 0.500 & 0.500 \\ 0.000 & 0.000 & 0.500 & 0.000 & 0.500 \\ 0.500 & 0.000 & 0.500 & 0.500 & 0.000 \end{pmatrix}$$

Upon calculation Pagerank matrix would be

$$I = \begin{pmatrix} 0.000 \\ 0.000 \\ 0.333 \\ 0.333 \\ 0.333 \end{pmatrix}$$

Here we can notice that the page rank for pages 1 and 2 are calculated as 0, which should not be the case because page 1 and 2 have links coming in or going out of them.

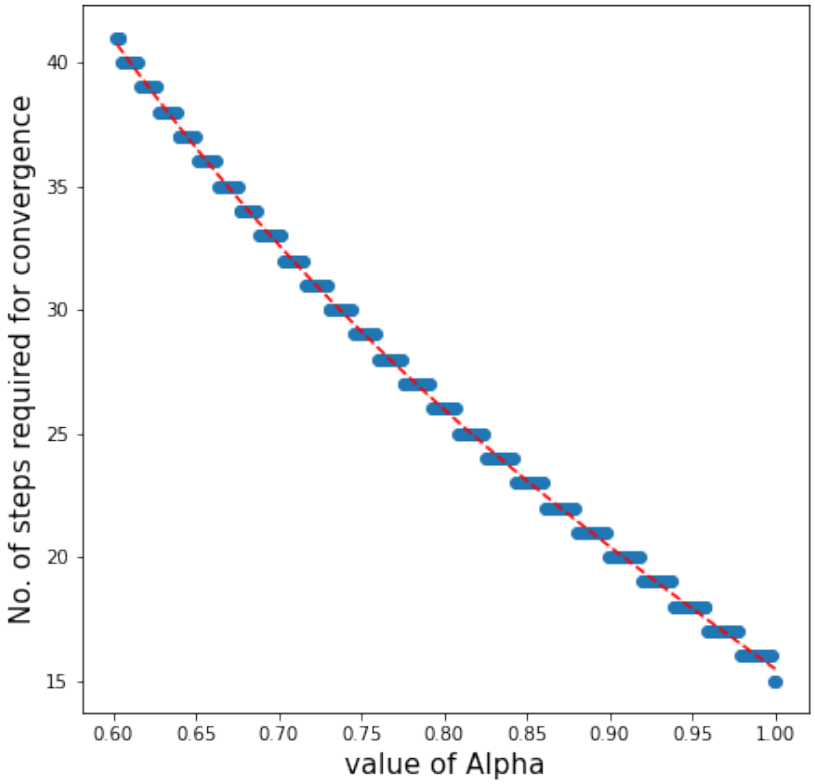
- Page 3, 4,5 make their own sub web of pages where no link is coming out from.
- This sub web is acting as "Rank Sink" making the ranks of page 1 and 2 0.
- To rectify this issue we use "Random Surfer" model.
- If a real Web surfer ever gets into a small loop of web pages, it is unlikely that the surfer will continue in the loop forever.Instead, the surfer will jump to some other page.
- The additional matrix E can be viewed as a way of modeling this behavior. All the elements of E are 1.

$$G = \alpha S + (1 - \alpha)(1/n)E$$

The corresponding Pagerank Matrix would be:

$$I = \begin{pmatrix} 0.052 \\ 0.052 \\ 0.304 \\ 0.288 \\ 0.304 \end{pmatrix}$$

Changing the value of α can change in convergence of power method. Generally the value of α is taken as 0.85. This is how the no. of steps required for convergence of the output pagerank matrix varies with no. of web pages:



5 Implementation in Python

5.1 Simple Pagerank Algorithm

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
G = nx.DiGraph()
G.add_edges_from(
[(1,2), (1,3), (2,4), (3,5), (3,2),
(4,5), (4,2), (4,6), (5,6),(5,7),(5,8),
(6,8),(7,5),(7,8),(7,1),(8,6),(8,7)])
nx.draw(G,with_labels=True)
plt.savefig('graph1.png', format='PNG')
plt.show()

l=list(G.edges())
n=[]
for i in l:
    if i[0] not in n:
        n.append(i[0])
```

```
    if i[1] not in n:
        n.append(i[1])
import numpy as np
H=np.zeros((len(n),len(n)))
d=dict.fromkeys(n)
for i in d:
    d[i]=[]
for i in d:
    for j in l:
        if i==j[0]:
            d[i].append(j)
import pandas as pd
for i in d:
    k=len(d[i])
    for j in d[i]:
        x=j[1]-1
        H[x][i-1]=(1/k)
print(pd.DataFrame(H,index=n,columns=n))

def pagerank_simple(H,m):
    I =np.zeros((len(H),1))
    I[0][0] = 1

    for j in range(0,m):
        I = np.dot(H,I)
    return I

print(pd.DataFrame(pagerank_simple(H,100),index=n))
```

5.2 Dangling Nodes

```
G1= nx.DiGraph()
G1.add_edges_from(
[(1,2), (1,3), (3,5), (3,2),
(4,5), (4,2), (4,6), (5,6),(5,7),(5,8),
(7,5),(7,8),(8,6),(8,7)])
nx.draw(G1,with_labels=True)
plt.savefig('graph2.png')
plt.show()

l1=list(G1.edges())
n1=[]
for i in l1:
    if i[0] not in n1:
        n1.append(i[0])
    if i[1] not in n1:
        n1.append(i[1])
import numpy as np
```

```
H1=np.zeros((len(n1),len(n1)))
d1=dict.fromkeys(n1)
for i in d1:
    d1[i]=[]
for i in d1:
    for j in l1:
        if i==j[0]:
            d1[i].append(j)
import pandas as pd
for i in d1:
    k=len(d1[i])
    for j in d1[i]:
        x=j[1]-1
        H1[x][i-1]=(1/k)
print(pd.DataFrame(H1,index=n1,columns=n1))

print(pd.DataFrame(np.around(pagerank_simple(H1,100),4),index=n1))
```

5.3 Updating H : Probabilistic Interpretation

```
result = np.all((H1 == 0), axis=0)
A=np.zeros((len(n1),len(n1)))
for i in range(len(result)):
    if result[i]:
        A[:,i]=1/(len(n1))

S=H1+A
print(pd.DataFrame(S,index=n1,columns=n1))
print(pd.DataFrame(pagerank_simple(S,100),index=n1))
```

5.4 Rank Sink

```
G2= nx.DiGraph()
G2.add_edges_from(
[(1,2), (1,5), (2,1), (2,3),(3,5),(3,4),
(4,5), (4,3), (5,4), (5,3)])
nx.draw(G2,with_labels=True)
plt.savefig('graph3.png')
plt.show()

l2=list(G2.edges())
n2=[]
for i in l2:
    if i[0] not in n2:
        n2.append(i[0])
    if i[1] not in n2:
        n2.append(i[1])
```

```
import numpy as np
H2=np.zeros((len(n2),len(n2)))
d2=dict.fromkeys(n2)
for i in d2:
    d2[i]=[]
for i in d2:
    for j in l2:
        if i==j[0]:
            d2[i].append(j)
import pandas as pd
for i in d2:
    k=len(d2[i])
    for j in d2[i]:
        x=j[1]-1
        H2[x][i-1]=(1/k)
print(pd.DataFrame(H2,index=n2,columns=n2))

result = np.all((H2 == 0), axis=0)
A1=np.zeros((len(n2),len(n2)))
for i in range(len(result)):
    if result[i]:
        A1[:,i]=1/(len(n2))

S1=H2+A1
print(pd.DataFrame(S1,index=n2,columns=n2))

print(pd.DataFrame(np.around(pagerank_simple(S1,100),4),index=n2))
```

5.5 Modified Page Rank Algorithm : Random Surfing

```
def pagerank(S,alpha,m):
    I = np.zeros((len(S),1))
    I[0][0] = 1
    J = np.ones((len(S),len(S)))
    G = (alpha*S)+(((1-alpha)/len(S))*J)
    for j in range(0,m):
        I = np.dot(G,I)
    return I

print(pd.DataFrame(np.around(pagerank(S1,0.85,100),4),index=n2))
```

5.6 Including Error Calculation for convergence

```
def pagerank_1(S,alpha=0.85,m=100,tol=0.0001):
    I = np.zeros((len(S),1))
    I[0][0] = 1
    J = np.ones((len(S),len(S)))
```

```
G = (alpha*S)+(((1-alpha)/len(S))*J)
for j in range(0,m):
    Ilast=I
    I = np.dot(G,I)
    err = sum([abs(I[n] - Ilast[n]) for n in range(len(I))])
    if err < len(I)*tol:
        print("converged after ",j," steps")
        return I
return I
```

References

[1] Goossens, M., Mittelbach, F., Samarin, *A LaTeX Companion*, Addison-Wesley, Reading, MA, 1994.

[2] Kopka, H., Daly P.W., *A Guide to LaTeX*, Addison-Wesley, Reading, MA, 1999.

[3] Lawrence Page and Sergey Brin and Rajeev Motwani and Terry Winograd, *The PageRank Citation Ranking: Bringing Order to the Web.*, Stanford InfoLab, 1999

[4] D.L.Isaacson and R.W. Madsen. *Markov Chains: Theory and Applications* John Wiley and Sons Inc., 1976