THE PAGERANK ALGORITHM

PROJECT SUBMITTED TO MAHATMA GANDHI UNIVERSITY

AS A PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE COMPLETION OF

THE DEGREE OF BACHELOR OF SCIENCE IN MATHEMATICS

By,
Pranav P
Reg No: 191106137
BSc. Mathematics

DEPARTMENT OF MATHEMATICS
CMS COLLEGE KOTTAYAM

DECLARATION

I do hereby declare that the project work entitled "The PageRank Algorithm" is a bonafide record of work carried out by me under the guidance of Ms. JOSMY ANN JOSE, Guest Faculty, Department of Mathematics, CMS College Kottayam.

Kottayam
31/03/2022

Pranav P
Sixth Semester
BSc. Mathematics
CMS College Kottayam (Autonomous)

CERTIFICATE

This is to certify that the dissertation entitled "The PageRank Algorithm" is a bonafide record of work done by Pranav P of CMS College, Kottayam under my guidance and supervision in partial fulfillment of degree in Bachelor of Science in Mathematics from Mahatma Gandhi University, Kottayam.

Kottayam
31/03/2022

Ms. Josmy Ann Jose
Guest Faculty
Department of Mathematics
CMS College Kottayam (Autonomous)

CERTIFICATE

This is to certify that the dissertation entitled "The PageRank Algorithm" is a bonafide piece of research work done by Pranav P of CMS College, Kottayam under the supervision of Ms. JOSMY ANN JOSE, Guest Faculty, Department of Mathematics C.M.S College, Kottayam in partial fulfillment of degree in Bachelor of Science in Mathematics from Mahatma Gandhi University, Kottayam.

Kottayam
31/03/2022

Dr. Ambily P Mathew
Head of the Department
Department of Mathematics
CMS College Kottayam (Autonomous)

ACKNOWLEDGEMENT

I wish to express my profound sense of gratitude and indebtedness of Ms. JOSMY ANN JOSE, Department of Mathematics CMS College, Kottayam for her kind hearted co-operation, valuable and inspiring guidance stimulating suggestions and encourage supervision.

I am very much grateful to Prof. Ambiliy P Mathew, Head of the Department of Mathematics for allowing me to use all facilities in the computer lab.

I extend my gratitude to all my friends for their suggestions and assistance during the preparation of this project.

Finally I would like to gratefully acknowledge the almighty for strength, grace and blessings received.

Kottayam
31/03/2022

Pranav P

INDEX

Introduction

The world wide web has an increasingly complex structure, which keeps on adding to its complexity day by day. Webpages in the order of hundreds of millions are being added year on year. Navigating this highly heterogeneous network is tiresome. Adding to the challenge, is the large number of inexperienced users and web pages which are designed to manipulate the searching algorithms.

But despite all the challenges the world wide web is not complete chaos and some order is in place. Unlike text documents each document in the world wide web is hypertext which provides us with some auxiliary information that is of utility. The web has an underlying link structure which can be utilized to navigate through it. The PageRank algorithm utilizes this structure to navigate the web with impressive accuracy. It is an intuitive way of ranking web pages, which formed the basis for Google’s web indexing algorithm during its early phase. It was invented by Larry Page one of the co-founders of google and it was built on top of years of research by his predecessors. The algorithm calculates the importance of a website as a function of its popularity. The algorithm outputs a probability distribution which represents the chances of random person surfing the web to arrive at any particular webpage.

Chapter – 1
Motivation behind PageRank

1.1 Birth of the internet

In the early 1960's even though computers were being used for research sharing of information with fellow researchers was still a dilemma. In order to share information with others researchers either had to travel to their location or have magnetic computer tapes created and sent to them via post or mail. There was great need for a technology that could make their networking easier. Internet was born out of this necessity, it was first started as a means for government researchers to share information with others. But it was restricted for public use. It was a proprietary technology of the US government.

During the cold war, when Russia launched the first artificial satellite, the US was pushed to create a network that would enable them to disseminate information even in the face of war. This led to the creation of ARPANET (Advanced Research Projects Agency Network), the predecessor to internet.

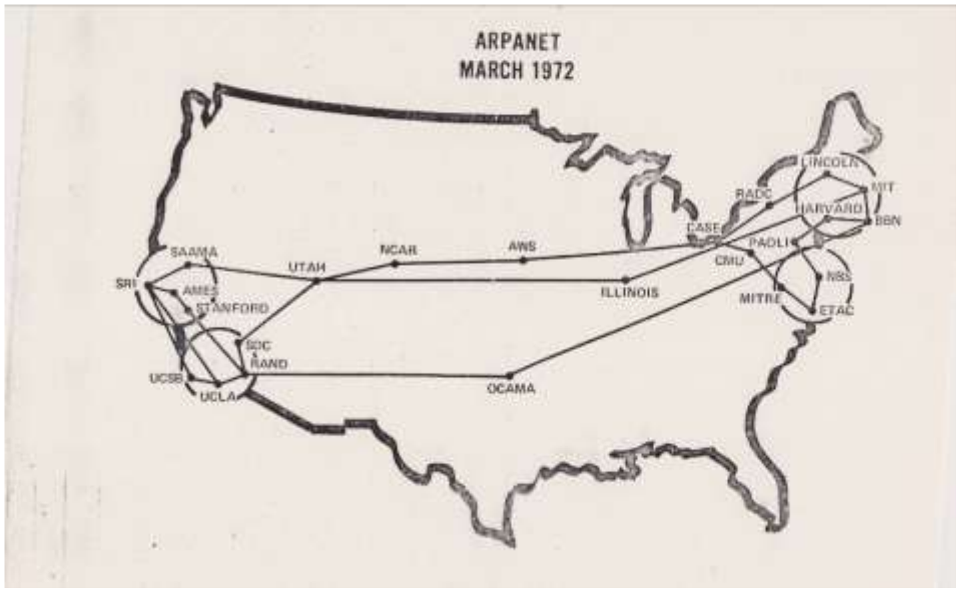


Fig 1.1 ARPANET Geographic Map

ARPANET was a success but its access was exclusive to government personals and certain specific research organizations. This led to the creation of numerous other networks in the US.

As new and new networks were being created cross communication between networks became very difficult. This called for the standardization of protocols for networks and communication. As a result a new communications protocol was created called the Transfer Control Protocol/ Internetwork Protocol (TCP/IP). With the advent of TCP protocol different networks were able to communicate with each other. Later ARPANET and other networks adopted TCP/IP and now every network was connected together by a common communication protocol which enabled them to communicate with each other. This data sharing environment was opened to the educational institutes in US. .com Internet service providers started operating in the late 1980's and the internet was fully commercialized by 1995.

1.2 Advent of world wide web.

The world wide web was invented by a scientist, Tim Berners Lee, while he was working in CERN. It was invented in 1989. It was originally designed to automate information sharing between scholars from different universities and other educational institutes. Web was built upon the existing data networks using hypertext and other technologies.

Tim Berners Lee, configured a computer at CERN to be the first web server. He had it up and running by the end of 1990. The first website contained information regarding the WWW project itself. It also contained literature on how to create and configure a web server. The URL of the first website was info.cern.ch. He also created a web browser for accessing websites. But the browser could only access static URLs since there wasn't a search engine at the time. So people had to remember the URLs of the sites they required, and could only search by URLs.

1.3 Early search engines

With the advent of world wide web attempts were made by developers across the world to create search engines. Because until then URL was necessary to access a website and people often had to keep directories of URLs.

The first major breakthrough in search engine development was “Archie” which made it possible to browse through a website’s directories. But you had to have a clue about the hierarchy of the website and where your required file was. This limitation was later surpassed by “Gopher”, which enabled users to search across a site’s directories to find your required file without having much knowledge about the website’s structure. Later numerous other search engines also came out, some of the major ones were Veronica, Compuserve and Genie. But all of them were short-lived.

In the late 1990’s search engines like The Virtual Library and Yahoo were created. But these weren’t actually “search engines”, these were just websites which contained human assembled catalogues of useful website links. In order to list your website in one of such directories people had to call them up and tell them the category, their websites belonged to.

“AltaVista” was the first search engine that actually functioned like a search engine. It was the first really fast search engine and it also covered much of the web. Along came other similar search engines like “AskJeeves”. But all of them were short-lived by the advent of Google and its PageRank algorithm. Google’s PageRank algorithm simply did a better job of searching the web much efficiently and made the overall experience much more user friendly.

But how did Google do this?. What was so special about the PageRank algorithm?

Chapter – 2
PageRank Algorithm

2.1 Related Work

The PageRank algorithm was built on top of years on research. One of the major ideas that contributed to the creation of PageRank algorithm was the link structure of the web.

Pitkow's Ph.D thesis on "Characterizing World Wide Web ecologies" included information on link analysis (Pitkow, 1997). Papers from Weiss and Spertus which talks about the various information that can be obtained from the link structure of the web and its applications (Weiss & Spertus, 1997) laid foundation to the invention of PageRank.

There were attempts to assign ranks to web pages using citation analysis utilizing the hypertextual citation structure of the web. The idea behind this was that if a lot of websites had backlinks to a particular website, then the later website should be important. For example, majority of the web pages nowadays has a backlink to some kind of a google service, which signifies its importance. This concept was utilized by scientists to create algorithms to rank websites based on their relative importance.

Now lets explore the link structure of the web in detail..

2.2 The link structure of the web

Despite the complex structure of the web there is still a way to structure the web utilizing the hypertext format of the web pages. This is because in web, a webpage can have hyperlinks to other webpages, in other words a webpage can cite another webpage.

For example consider a website C, C may have hyperlinks to other websites, these links are called forward links. Also there could be other webpages that have hyperlinks pointing to C, these links are called backlinks (Fig 2.1).

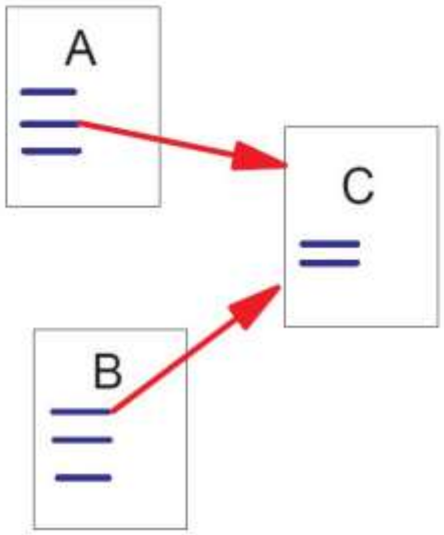


Fig 2.1: A and B are backlinks of C

Now consider any website, there are two possible cases.

1. **The website has a large number of backlinks**
   If a website has a large number of backlinks then it could signify that it is an important website. In fact in the early days of search engines the number of backlinks was used as a measure to predict the next nobel prize winner.

2. **The website has only a few backlinks**
   This is where it gets interesting, here we have further two cases.
   i. **It has a backlink from an important website.**
      This could imply that the website is also important
   ii. **It only has backlinks from some obscure websites.**
      This can imply that the website is not so important.

Now we know that counting just the number of backlinks is not sufficient to calculate the importance of a website. The nature of the webpages hosting those backlinks should also be considered. PageRank tries to approximate the importance of a webpage using the link structure taking all these factors into account.

2.3 Calculation of PageRank

Let x be a webpage. Let F_x be the set of pages that x point to and let B_x be the set of all pages that has a link to x ie the set of all webpages having backlink to x. Let $N_x = |F_x|$, ie the number of distinct forward links in x. Let k be a factor used for normalization so that the sum of ranks of all webpages is a constant.

We now define a much more simplified rank $R(x)$ of x. By

$$R(x) = \sum_{u \in F_x} \frac{R(u)}{N_x} \tag{1}$$

So we now defined the rank of a web page as a function of the backlinks to it as per our intuition. A diagrammatic representation of the calculation is given below

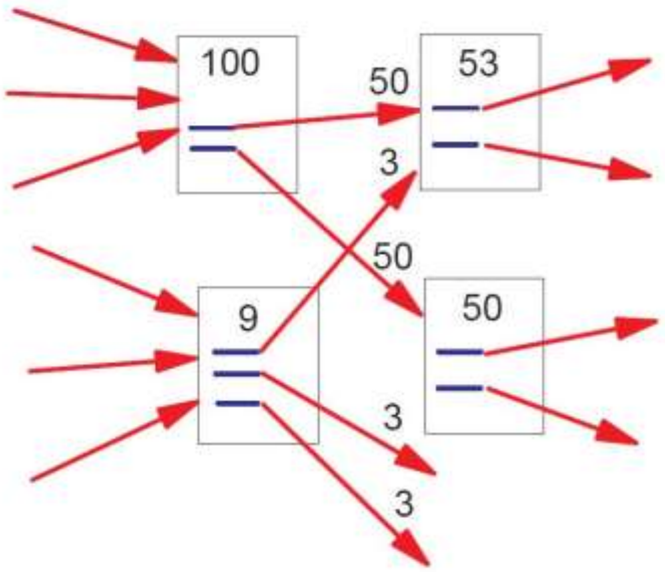


Fig 2.2: Propagation of rank across pages

As we can see from the above figure the rank of a website is evenly distributed among its forward links. The equation described above is a recursive equation, and initially, if there are N websites the initial rank is taken as $\frac{1}{N}$. But on successive iterations the rank of webpages will converge to a particular value and this is taken as their rank.

For instance consider the web graph of six webpages given below.

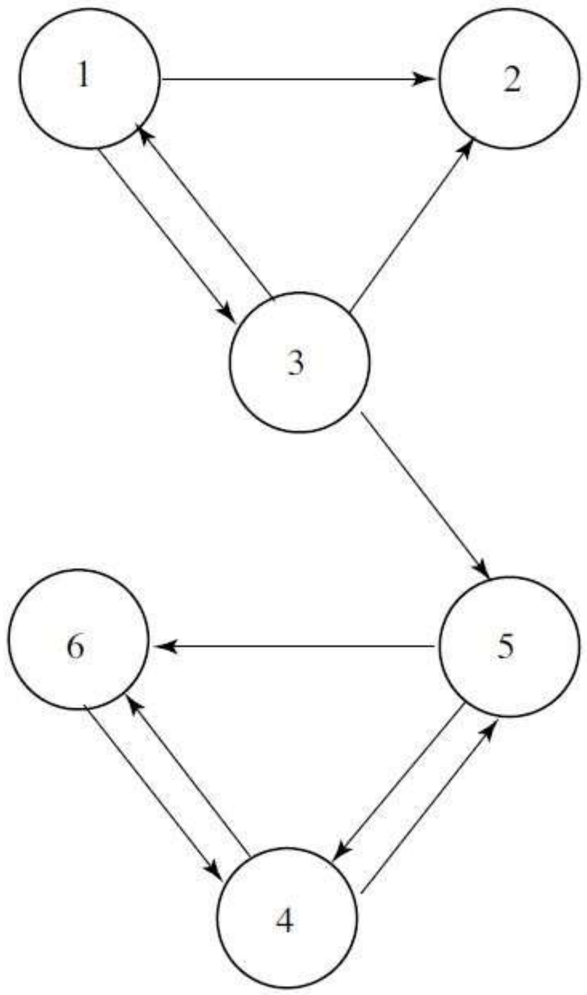


Fig 2.3

Now using the above mentioned the equation if we try to calculate the rank of these webpages. The result on each iteration is as follows.

Iteration 0	Iteration 1	Iteration 2	Rank at Iter. 2
$r_0(P_1) = 1/6$	$r_1(P_1) = 1/18$	$r_2(P_1) = 1/36$	5
$r_0(P_2) = 1/6$	$r_1(P_2) = 5/36$	$r_2(P_2) = 1/18$	4
$r_0(P_3) = 1/6$	$r_1(P_3) = 1/12$	$r_2(P_3) = 1/36$	5
$r_0(P_4) = 1/6$	$r_1(P_4) = 1/4$	$r_2(P_4) = 17/72$	1
$r_0(P_5) = 1/6$	$r_1(P_5) = 5/36$	$r_2(P_5) = 11/72$	3
$r_0(P_6) = 1/6$	$r_1(P_6) = 1/6$	$r_2(P_6) = 14/72$	2

2.4 Matrix Representation of the summation equations

Eq. 1 can only calculate the PageRank of one page at a time. But we can uses matrices to calculate a PageRank vector which is of dimension $n \times 1$ which gives the PageRank of all pages.

Using matrices, if you recall the link structure of the web talked earlier. We know that we can model the web as a graph. Let H be the adjacency matrix of the said graph.

$$H_{u,v} = \begin{cases} \frac{1}{N_u}, & if\ u\ is\ connected\ to\ v \\ 0, & otherwise \end{cases}$$

So H will be of dimension $n \times n$. Also let the PageRank vector be π^T. The H matrix for the web graph in Fig 2.3 is given below.

$$H = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

In the matrix we can see that the i^{th} row corresponds to the forward links in page i and the i^{th} column corresponds to the backlinks of page i. Now we introduce the PageRank vector at the k^{th} iteration, $\pi^{(k)T}$. Now using matrices, we can rewrite equation 1 as

$$\pi^{(k+1)T} = \pi^{(k)T} H \tag{2}$$

With $\pi^{(0)T} = \frac{1}{n} e^T$ where e^T is a vector of all ones. We can easily verify that equation (2) yields the same results as equation 1.

Now we can make some important observations from equation (2).

We can clearly see that the computation has time complexity of $O(n^2)$ where n is the size of the H matrix. But H is a *very sparse matrix.* ie most of the elements in H are zero. Now since H is a sparse matrix the time complexity of the computation can be reduced dramatically. In fact the actual time complexity of the computation is $O\big(nnz(H)\big)$ where $nnz(H)$ is the number of non-zero entries in H. So say that, in n websites each websites has on average 10 forward links, which means that H has $10n$ non zeros. But this computation isn't of quadratic time complexity, we can see that the time complexity is in fact linear, ie $O(n)$. This means that equation (2) only has a time complexity of $O(n)$.

In equation (2) the iterative process is nothing but the classical power method being applied to H.

Also H looks a lot like a stochastic transition probability matrix in a markov chain (More on this later). But the rows corresponding to dangling nodes are zero rows. And all the other rows which correspond to non-dangling nodes are stochastic. So H can be said to be a substochastic matrix.

Dangling Links: In the web graph dangling links are links to webpages which does not have any outgoing links. Dangling links poses a problem to our model since it is not clear where their weight should be distributed.

2.5 Problems with the initial model

This simplified PageRank has a problem. Imagine the scenario when there are two webpages and say both of them are pointing to each other. Now consider another website pointing to one of these websites. When we run the current algorithm the rank accumulates in the loop with the two websites and is not distributed to the third one because the third one doesn't have any inedges. This loop forms a sort of a trap that locks the rank inside, it is called rank sink (Fig 2.4).

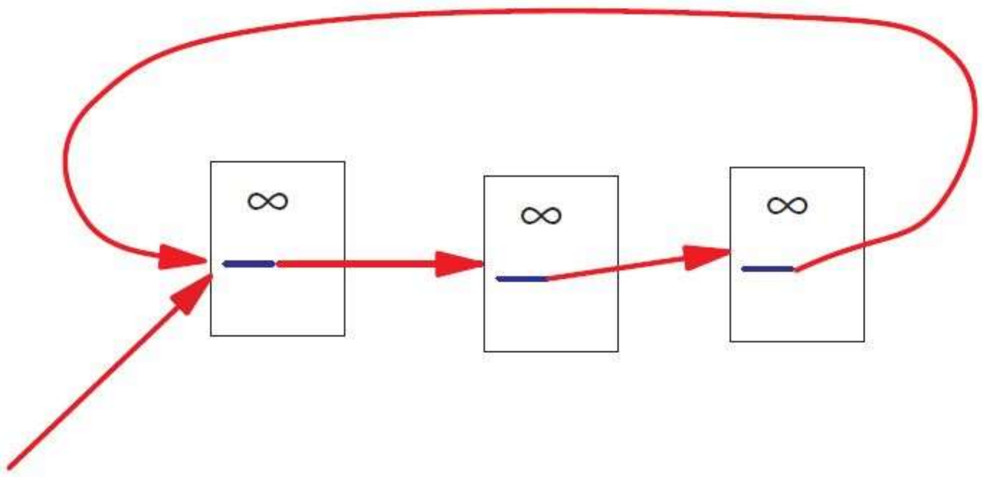


Fig 2.4: Rank sink

For instance in Fig 2.3 the cluster of nodes 4, 5 and 6 form a rank sink which hoards the rank. So after 13 iterations of equation (2) we get

$$\pi^{(13)T} = [\quad 0 \quad 0 \quad 0 \quad 2/3 \quad 1/3 \quad 1/5 \quad]$$

But this is actually wrong. Since the rank of a webpage being zero means the probability that a user lands on that webpage is nil, but even if a user can't reach a webpage by following hyperlinks from other websites, he can always go directly to that webpage using its URL. This should also be accounted. Ideally we want the rank of all webpages to be a positive value. Here, due to the formation of a rank sink, the rank sink accumulates a lion's share of the ranks.

We have already seen that equation (2) is very similar to power method applied to a Markov matrix. The Markov matrix theory is well developed so we can use it to make necessary adjustments to our matrix, so that it can account for the issues mentioned above and to insure that on applying power method the matrix converges. For Markov matrices we know that, if P is a Markov matrix, then for any starting vector, on application of power method the Markov matrix P converges to a unique positive vector called stationary vector as long as P is stochastic, irreducible and aperiodic. So the problems to the PageRank calculation caused by sinks can be solved if we can make some adjustments to H to make it a Markov matrix with the desired properties.

Note : If a matrix is both irreducible and aperiodic then it is said to be primitive.

Now to solve the above mentioned problems we introduce another intuitive model or PageRank algorithm.

2.6 The Random Surfer Model

The PageRank algorithm is also based on an intuitive model. PageRank can also be thought as the probability of a random surfer, who is searching the web, to reach a particular page after a series of clicks. For example, say the normalized PageRank of a webpage is $.6$ which means that if a person starts surfing the web randomly ie he is just going into webpages and randomly clicking hyperlinks. Then the probability that he is going to land on the said webpage is .6 .

When looking at this from mathematics point of view, we can see that this is nothing but a random walk on the web graph, ie that graph describing the entire web with webpages as vertices and hyperlinks as directed edges.

2.7 Making adjustments to the initial model

Now when we consider the Random Surfer model of PageRank, imagine a surfer randomly bouncing around in the web. He reaches a website, chooses a hyperlink at random and then proceeds to the webpage corresponding to that hyperlink. In the long run the proportion of time he spends in a particular webpage determines the importance of that webpage. Because if he spends a large amount of time on a particular webpage then it means that, in bouncing across the web randomly, he found himself reaching that website more frequently. Pages that he revisits frequently are also important because it means that there are more backlinks to that webpage.

Occasionally he may run into some problems. He may reach a dangling node, ie a node with no forward links. To fix this we do our first adjustment known as ***stochasticity adjustment*** where the zero rows of H, ie the rows corresponding to the dangling nodes, are replaced by $\frac{1}{n}e^T$, which makes H a stochastic matrix. What is actually happening here is that when the surfer reaches a dangling node, he would move to a random webpage using its URL. So the probability in this case would be $\frac{1}{n}$ because there are equal chances for the surfer to choose any webpage.

After making stochasticity adjustment for the H matrix of the web graph Fig 2.3. Let the new stochastic matrix obtained be S. Then S is given by

$$S = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Writing this stochasticity fix mathematically we see that S if created from a rank one update to H. ie we can write S as,

$$S = H + a\left(\frac{1}{n}\, e^T\right)$$

where a is given by,

$$a_i = \begin{cases} 1, & if\ page\ i\ is\ a\ dangling\ node \\ 0, & otherwise \end{cases}$$

The binary vector a is called a ***dangling node vector.*** Therefore S is a combination of H and a rank one matrix $\frac{1}{n}\, e^T$.

Stochasticity adjustment assures that the matrix obtained after adjustment, S, is stochastic. But stochasticity along cannot guarantee the convergence of the matrix. For this the matrix should be aperiodic as well as irreducible. To ensure that further adjustment is needed. This adjustment is called **primitivity adjustment.** With this adjustment the resultant matrix would be stochastic and primitive. And hence a stationary matrix can be obtained by power iteration.

To explain primitivity adjustment let us recall the random surfer model. Even though the surfer bounces around the web utilizing the hyperlink structure of the web at times the surfer might get bored and could enter a random URL and "teleport" to an entirely new webpage. After teleportation he continues to follow the hyperlink structure and bounce around. This was not accounted in the initial model. So to model this activity mathematically we introduce a new matrix G which is given by

$$G = \alpha S + \frac{(1-\alpha)ee^T}{n}$$

where α is a scalar between 0 and 1. G is called the Google matrix. In this equation α is a parameter that determines the proportion of time a random surfer follows hyperlinks as opposed to teleporting. For instance if $\alpha = .4$ then 50% of the time the surfer follows the hyperlink structure and 60% of the time, he teleports to a random webpage. The teleportation is random because the matrix corresponding to the teleportation $\frac{1}{n}ee^T$ is a $n \times n$ matrix having all values as $\frac{1}{n}$. So the chances that the teleporter teleports to any site is equal. For practical implications we introduce another vertex to the web graph called supernode. Which has a bidirectional edge to all vertices. So by teleportation we mean that the surfer is going to the supernode and then moving to the webpage of his requirement.

So the final equation for PageRank calculation can be given as

$$\pi^{(k+1)T} = \pi^{(k)T}G$$

which is simply the power method applied to G.

Now if we apply this equation to the web graph in Fig 2.3 with $\alpha = .9$. The stochastic, primitive matrix G would be,

$$G = .9H + \left(.9\begin{pmatrix}0\\1\\0\\0\\0\\0\end{pmatrix} + .1\begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix} \right) 1/6\,(1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1)$$

$$= \begin{pmatrix} 1/60 & 7/15 & 7/15 & 1/60 & 1/60 & 1/60 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 19/60 & 19/60 & 1/60 & 1/60 & 19/60 & 1/60 \\ 1/60 & 1/60 & 1/60 & 1/60 & 7/15 & 7/15 \\ 1/60 & 1/60 & 1/60 & 7/15 & 1/60 & 7/15 \\ 1/60 & 1/60 & 1/60 & 11/12 & 1/60 & 1/60 \end{pmatrix}$$

and on applying the equation, we get the stationary matrix as,

$$\pi^T = (.03721 \quad .05396 \quad .04151 \quad .3751 \quad .206 \quad .2862)$$

this can be interpreted as, 3.721% of time the random surfer visits webpage 1. Hence the pages in the web graph Fig 2.3 can be ranked as

$$(4 \quad 6 \quad 5 \quad 2 \quad 3 \quad 1)$$

which means that the 4^{th} website is the most important one and the 1^{st} one is the least important one as per the algorithm.

PageRank also solves another problem that many algorithms faced. Many algorithms used the number of backlinks as a measure to calculate the popularity of a website. But this posed a problem since people could easily manipulate the algorithm. For instance if someone wanted to make their webpage more popular they could easily spam dozens of webpages having backlinks to the said webpage.
But in the PageRank algorithm, even though there are a large number of webpages pointing to it, those webpages don't carry much weight on their own since no webpage points to them. So even though the number of backlinks is high, their weight would be significantly less. There was another concern, ie people could increase the rank of their webpage by paying other webpages to have a backlink pointing to them. But this was considered reasonable since one had to pay money to accomplish this.

Chapter – 3
Crawlers and Markov Chains

3.1 Storing and using URLs

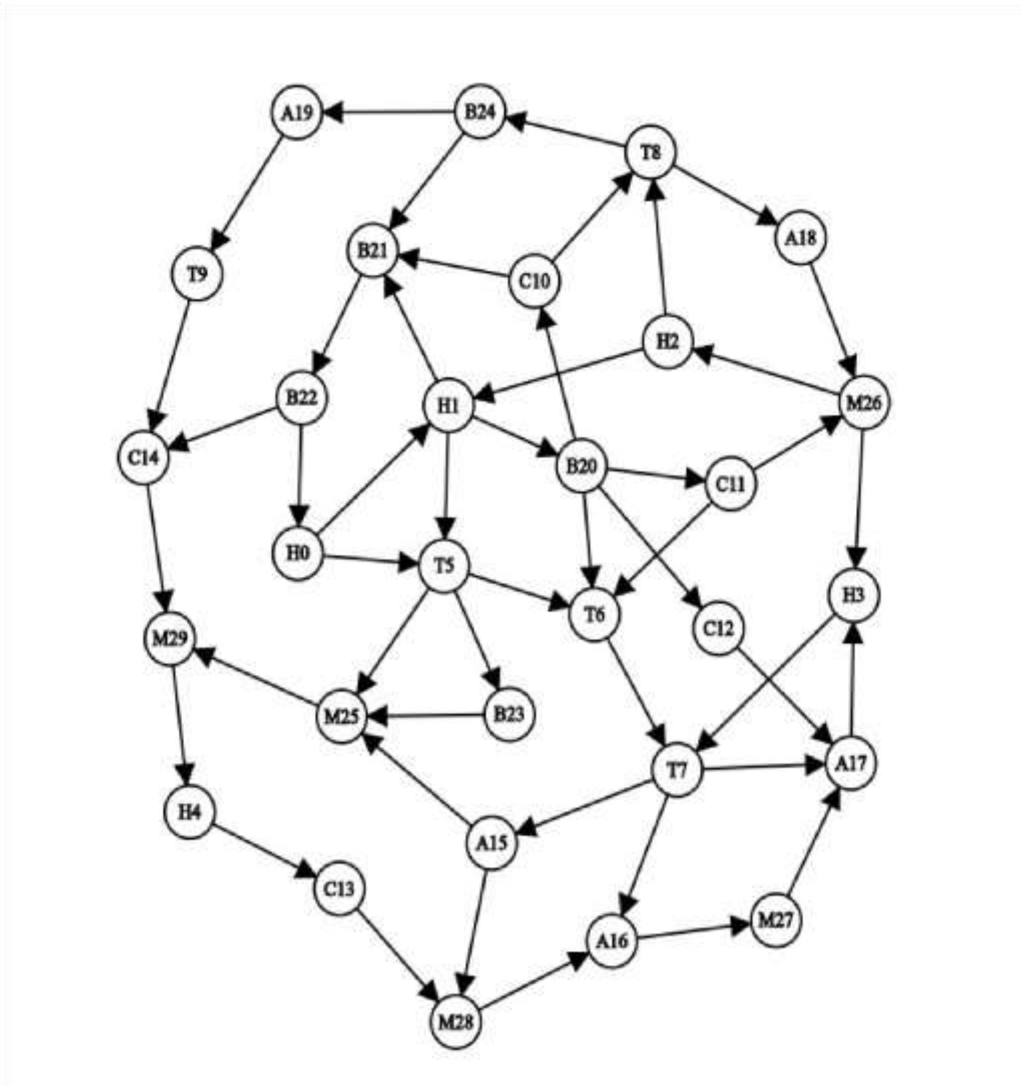
We first of all convert each URL to an integer ID and store them in the database. We can now sort these in accordance with their parent IDs and remove the dangling links from the link database (A few iterations are sufficient to remove a vast majority of them). Then an initial assignment of ranks are provided (We have a wide variety of strategies to do so). Here, the thing we should understand is that if we are to iterate till convergence, the final value is not dependent on the initial values but the rate of convergence is. So, a proper assignment of initial values can really lead us to a faster convergence with only a feasible finite number of iterations being executed.

For the weights of every page, memory is allocated. The size of RAM required differs with the precision of the floating point used. If sufficient RAM is not available, multiple passes can be made. The weights from the current time execution are kept in memory and the previous ones are accessed from the disk, linearly. Since the link database is sorted, access to it is linear. Once the links are converged we add in the dangling links which were removed earlier and then recompute the ranking. This is to ensure that the dangling links have a non-zero weight. With better optimization and not so strict convergence criteria the calculations can be made faster. Usage of efficient techniques for the estimation of eigenvectors could also be used to improve performance. The thing to note here is that the cost required to compute the PageRank is not that significant compared to the building cost of full text index.

3.2 Web Crawlers

A web crawler, spider or search engine bot is a program that downloads and indexes contents present in the internet. In our implementation of the PageRank the use of the spider (Web Crawler) is to find all of the links present in the internet. Since it is practically impossible to use our personal computers to index the whole world wide web, we created a virtual internet space which contains 30 websites having different contents. The spider is first asked to look at a few websites of which the URLs are already known. The spider will then scrape the information in each website and will find the list of all hyperlinks to other websites from the current web page. The spider will then move to one of the websites at random. The spider executes a random walk from website to website. When the spider reaches a website which it has not visited below it adds the URL of the website to the database. Thus the spider will be able to retrieve almost all of the URLs present in the internet.

We used Python along with Django to power up this so-called virtual internet space. We wrote in 30 HTML files each representing a static web page and gave each of them a static URL. Each of the HTML files contain hyperlinks to other HTML files.



The above diagram represents our virtual internet space. Here each vertex is a web page and each directed edge is hyper reference from one web page to another(The head of the arrow indicates the web page to which the hyperlink points and the tail indicates the web page from which the hyperlink points). Our spider makes a random walk from one web page to another by following the arrows.

In Python we use a library called *scrapy* to write our own spider. We named it Oswald. Oswald, just like a normal spider would start off with 6 known URLs(namely h0, t5, c10, a15, b20 and m25 in the above diagram) and would randomly move through web pages. As it moves through web pages it uses a list called *collected_urls* to store URLs on the go. When it reaches a new web page it cross checks with the URLs present in the *collected_urls* and appends it to *collected_urls* if it is not already present. Even before Oswald starts adding URLs to *collected_urls* we append the URLs already present in the database if there were any (Something like that could happen because this might not be the first time Oswald crawls through the web. The result of the previous surfing could already be present in the database).

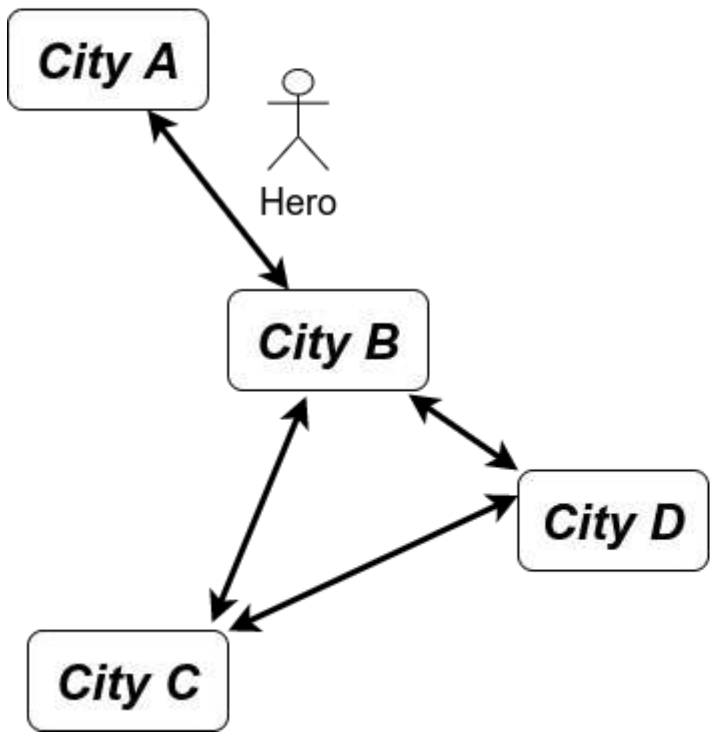
The code used for the implementation of Oswald is given below.

```
import scrapy
import random


class OswaldSpider(scrapy.Spider):
    name = 'oswald'
    count = 0
    start_urls = [
        'http://127.0.0.1:8000/health/h0',
        'http://127.0.0.1:8000/technology/t5',
        'http://127.0.0.1:8000/cars/c10',
        'http://127.0.0.1:8000/animals/a15',
        'http://127.0.0.1:8000/business/b20',
        'http://127.0.0.1:8000/movies/m25',
    ]
    collected_urls = list()
    prev_url = start_urls[0]

    def parse(self, response):
        next_page_url = ''
        for link in response.css('a'):
            next_page_url = link.attrib['href']
            if next_page_url not in self.collected_urls:
                self.count += 1
                self.collected_urls.append(next_page_url)
                if next_page_url not in self.start_urls:
                    self.start_urls.append(next_page_url)
                yield {
                    # self.prev_url: next_page_url,
                    next_page_url: self.count
                }
        # self.prev_url = next_page_url
        # yield response.follow(next_page_url, callback=self.parse
        urls = response.css('a::attr(href)').getall()
        random.shuffle(urls)
        next_page_url = urls[0]
        yield scrapy.Request(next_page_url)
```

3.3 Markov Chain

A *Markov Chain* is a sequence of events where the probabilities of the future ONLY depend on the present.

A more simplified explanation can be given using an example. Assume that we have a collection of cities each connected via some rail connection. Not all cities are connected to each other. Now, we have a Markovian superhero who moves from one city to another to solve problems in each of them. Now our superhero does have a few specialties. One is that the superhero cannot fly. That is he has to take the train to go from one city to the other. The other specialty is that he is a bit of a forgetful one. That is, he forgets where has had been in the past. So, if the superhero moved from one city to another and wants to go to another, then what he does is, he looks at all the possible routes to which he can travel and then randomly picks one. It goes as in the following diagram.



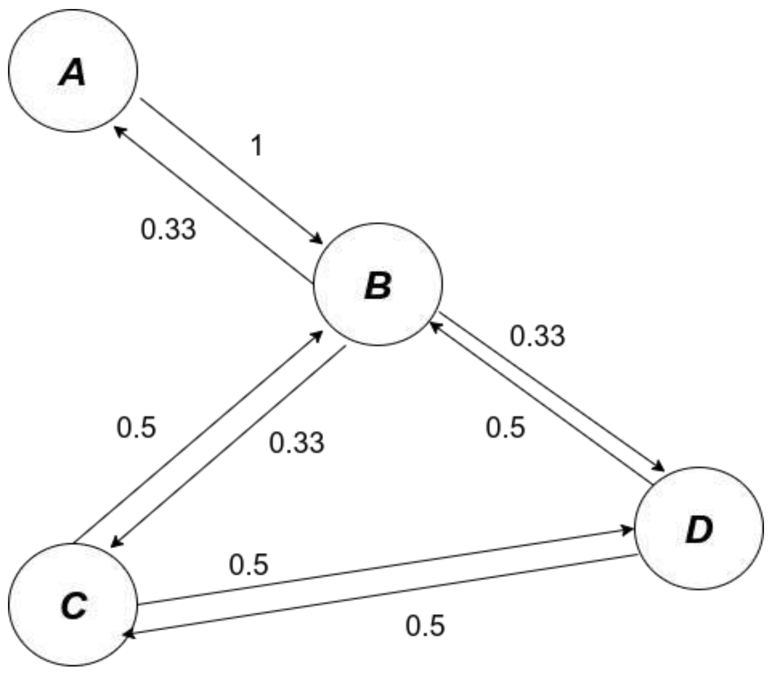
Now if the superhero is at City B, then it has to choose between 3 possibilities which are City A, City C and City D. If the superhero was at City D, it would have 2 possibilities which are

City B and City C. In each case, our Markovian superhero chooses these between these options at random. That is, this superhero chooses its next destination randomly based on where they currently are. That is the probability of each next destination depends only on the present position. Hence, the chain of these events is a Markov chain.

We've a lot of real life scenarios of Markov chains. As an example, in the game of snakes and ladder each move is decided entirely based on the present throw of the dice, hence a Markov chain.

Now we define something called a *Transition Diagram.* A Transition Diagram is a flowchart in which each box is drawn as circles and are called states and each edge connecting the states would be assigned a weight.

Here in our Markovian superheroes example, we would get a Transition Diagram simply by replacing each and every rounded rectangular boxes with a circle and by assigning to each edge (Since, in our diagram, our edges were two sided, we split them into two unidirectional edges which move both from and to a state), the probability with which the route is chosen to move from the city at the tail of the edge to the head of the edge.



The above diagram illustrates the transition diagram of our Markovian superhero's example.

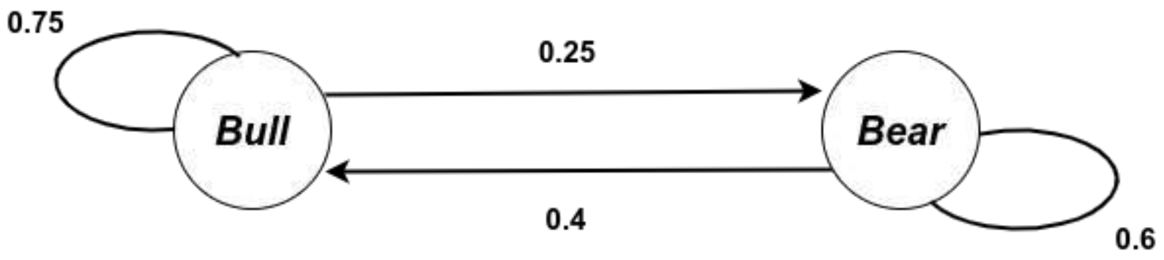
Now we illustrate another example, a stock market example, so that we can move on to the next topic with ease.

Now consider this example of a stock market where after each week

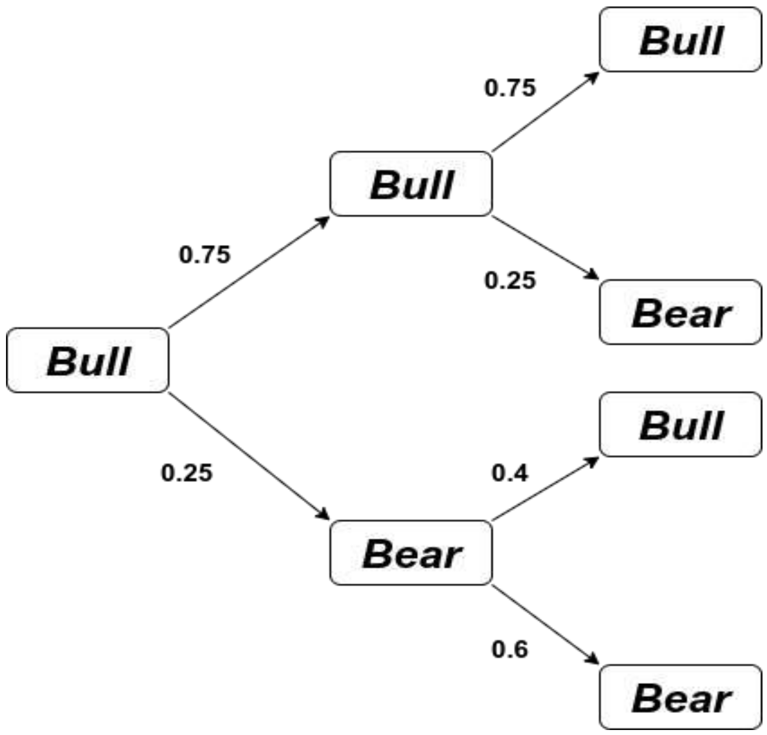
- 75% of the time a bull market is followed by a bull market
- 60% of the time a bear market is followed by a bear market

(If the stack market goes up, we call it a bull and if it goes down we call it a bear)

The transition diagram of the example goes as follows.



Now we can easily make prediction for the future using this diagram. For example, if it is a bull market this week, then the probabilities in two weeks is figured using the following diagram.



From the diagram it is clear that the probability for a bull after two weeks can be attained in two ways. That is either as

- Bull → Bull → Bull
- Bull → Bear → Bull

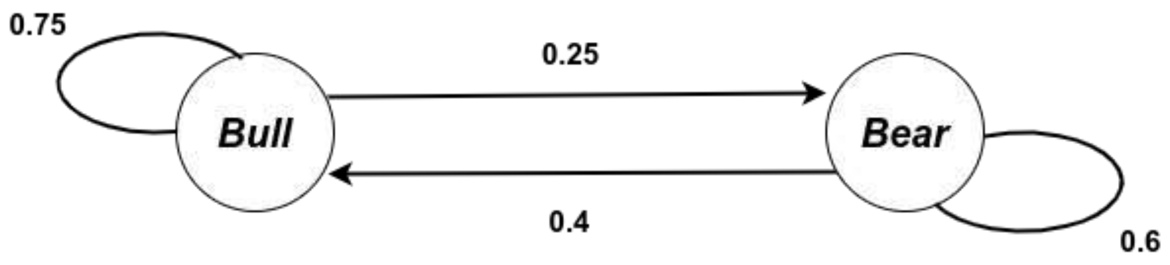
So, the total probability would be the sum for each of them.

$ie, 0.75 \times 0.75 + 0.25 \times 0.4 = 0.66$

With this method we can predict 100 or so weeks into the future. But this is a cumbersome task to do. This is where we use another tool at our disposal.

3.4 Transition Matrix

Imagine that we've a transition diagram(Let it be the one mentioned in the previous stock market example).



Now, in our problem we've had the idea that the current week is a bull week. So, we set an initial vector S_0 to a value which indicates the current state of the system. This vector S_0 is called the initial vector.

$$S_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

That is, S_0 shows that the probability of a bull is 100% while that of a bear is 0%.

Now we take in a matrix which is indicative of the weights of the edges in our transition diagram. It is the matrix P.

$$P = \begin{array}{c|cc} & Bull & Bear \\ \hline Bull & 0.75 & 0.4 \\ Bear & 0.25 & 0.6 \end{array}$$

Now this matrix is called a *Transition Matrix, Markov Matrix* or *Stochastic Matrix.* Now a stochastic matrix is a square matrix used to describe the transitions of a Markov chain. Each of its entries is a non-negative real number representing a probability.

Now, using this matrix P we can make calculations into the future much more easily. That is, let S_1 be the state of the market after 1 week. Then we have this relation $S_1 = PS_0$.

$$S_1 = PS_0$$

$$= \begin{bmatrix} 0.75 & 0.4 \\ 0.25 & 0.6 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix}$$

Similarly if S_2 is the state of the market after the 2nd week, then we will still have the relation $S_2 = PS_1$.

But

$$S_1 = PS_0$$
$$S_2 = P \times PS_0$$
$$S_2 = P^2 S_0$$

So,

Calculations show that $S_2 = \begin{bmatrix} 0.66 \\ 0.34 \end{bmatrix}$ ie, the probability for a bull in the 2nd week is 0.66, exactly the same value we had got while proceeding through the transition diagram. But in this method this are a bit more generalized and the state of the market after *n* weeks can be calculated by the formula.

$$S_n = P^n S_0$$

3.5 Convergence of matrices

We introduce a new theorem and a method which will be highly useful for our further proceedings.

Theorem. *If M is a Markov matrix, there exists a $\vec{x} \neq \vec{0}$ such that $M\vec{x} = \vec{x}$.*

Proof. The determinants of a square matrix and its transpose are identical. This means that their characteristic polynomial are identical, which in turn means that they have the same eigenvalues. When left-multiply a matrix by a vector, the result is a linear combination of the matrix rows. In particular, left-multiplying by a vector of all 1s sums the rows of the matrix. Each column of a Markov matrix sums to 1, therefor $\mathbf{1}^T M = \mathbf{1}^T$. Transposing, we see that $\mathbf{1}$ is an eigenvector of M^T with eigenvalue 1, therefore 1 is an eigenvalue of M and there must by definition exist some non-zero vector $\vec{x}$ such that $M\vec{x} = \vec{x}$ ☐

Now our objective is to find such an eigenvector as mentioned in the above theorem. For that we use the power method. The power method is an iterative algorithm, here used to find an eigenvector $\vec{x}$ such that $M\vec{x} = \vec{x}$. The algorithm is as follows.

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

Now using this algorithm, we find such an eigenvector as mentioned in the Theorem at the start of this section.

Chapter 4
Implementing the algorithm

4.1 Placing the theory into practice

In our last chapter we've discussed a few mathematical concepts. We had already given a footnote on how these mathematical tools would be helpful in solving our problem. But just for a brief review, we understand that our random surfer (A person who surfs over the internet just clicking hyperlinks in one website, redirects to the other then randomly clicks another hyperlink in that website and moves on to the other and so forth) would have a higher probability to end up in some websites than the others. These websites are the ones which are referred more by other sites and hence the more popular ones. Now in order to find the probability of the random surfer to be at a particular website after n click, we could use the idea of the Markov matrix and that of the power method.

Now we implement what we have been talking. First of all, we use our beloved Oswald(spider) to fetch all the web addresses, then remove the dangling nodes and then using it we generate a transition matrix.

```
def probability_calc(links):
    prob_vector = [0] * len(universal_link_set)
    try:
        prob_value = 1/(len(links))
        for link in links:
            prob_vector[universal_link_set[link]] = prob_value
        return np.array(prob_vector)
    except ZeroDivisionError:
        return prob_vector

def generate_transitional_matrix():
    transitional_space = []
    for link in universal_link_set.keys():
        extracted_link_list = extract_links(link)
        prob = probability_calc(extracted_link_list)
        transitional_space.append(prob)
    return np.column_stack(transitional_space)
```

Here the function *probability_calc* generates a vector having dimension equaling the number of websites where each entry is either *0* if there is no hyperlink to the web page at the concerned index or the reciprocal of the number of hyperlinks from the website. In the function *generate_transitional_matrix* all we do is stack up all these column vector to form our transition matrix.

Our initial state vector is generated by taking a column vector of dimension equal to the number of web pages in the internet (here our virtual internet space) where each entry has a value equal to the reciprocal of the number of web pages in the internet (Since the random surfer can type in the URL of any web page to start off with).

Now all we need to do is to pre multiply the initial state vector with the transition matrix again and again until it converges. But before that we add the damping factor to the matrix and then use the power method to get the concerned eigenvector.

The entire code for this process is given below.

```
from bs4 import BeautifulSoup
from urllib.request import Request, urlopen
import numpy as np
import re

def universal_link_set_retriever():
    with open('sites.txt', 'r') as f:
        link_set = f.read().strip().split()
        link_set.sort(key=lambda x: x.split('/')[-1])
        link_dict = dict(zip(link_set, [x for x in range(30)]))
        return link_dict

universal_link_set = universal_link_set_retriever()

def extract_links(url):
    req = Request(url) # "http://127.0.0.1:8000/cats/1"
    html_page = urlopen(req)

    soup = BeautifulSoup(html_page, "lxml")

    links = set()
    for link in soup.findAll('a'):
        hyperlink = link.get('href')
        if (hyperlink is not None) and (hyperlink != url):
            links.add(hyperlink)

    #print(links)
    return links

def probability_calc(links):
    prob_vector = [0] * len(universal_link_set)
    try:
        prob_value = 1/(len(links))
        for link in links:
            prob_vector[universal_link_set[link]] = prob_value
        return np.array(prob_vector)
    except ZeroDivisionError: #To address cases where a website has no hyperlinks
        return prob_vector
```

```
def generate_transitional_matrix():
    transitional_space = []
    for link in universal_link_set.keys():
        extracted_link_list = extract_links(link)
        prob = probability_calc(extracted_link_list)
        transitional_space.append(prob)
    return np.column_stack(transitional_space)

def add_damping_factor(P):
    beta = 0.85
    n = len(universal_link_set)
    v = np.array([1] * n).T
    e = np.array([1/n] * n)

    #R = beta*P + (1-beta)*(np.dot(v, e.T))
    R = beta*P + (1-beta) * 1/n
    return R

def generate_converged_rank_vector(transitional_matrix):
    size = transitional_matrix.shape[0]
    initial_rank = np.ones(size) * 1/size
    previous_ranking = initial_rank
    iterations = 1
    new_rank = transitional_matrix @ initial_rank

    while np.linalg.norm(previous_ranking - new_rank) > 1e-4:
        previous_ranking = new_rank
        new_rank = transitional_matrix @ new_rank
        iterations += 1

    return new_rank, iterations

if __name__ == '__main__':
    transitional_matrix = generate_transitional_matrix()
    new_transitional_matrix = add_damping_factor(transitional_matrix)
    rank = generate_converged_rank_vector(new_transitional_matrix)
    print(rank)
```

4.2 term frequency – inverse document frequency

In order to finish our job, we need one more algorithm which relates to the keywords we use to make a search. For that, here in our demonstration program we use the tf-idf algorithm (term frequency – inverse document frequency). Tf-idf was one of the old algorithms used in search engines prior to Google. The algorithm is nothing but a simple equation used to calculate a weight for each web page based on the keyword used to make the search.

If *t* is the term(word) used for search, *d* is the document in which the search is made and *N* is the count of the corpus where corpus is the total document set, then term frequency *tf* is

$$tf(t,d) = \text{count of } t \text{ in } d/\text{number of words in } d$$

and document frequency *df* is

$$df(t) = \text{occurrence of } t \text{ in } N \text{ documents}$$

Now, inverse document frequency is just

$$idf(t) = N/df$$

We use the log function to dampen the effect of a large *N*

$$idf(t) = \log(N/(df+1))$$

The addition of the number 1 is to avoid the log value being 0 when *N = df*

Now we show the code used to implement this

```
from math import log

import os
import urllib.request
from bs4 import BeautifulSoup


def term_frequency(term, doc):
    doc_string = doc.lower()
    term_reps = doc_string.count(term.lower())
    len_of_document = float(len(doc_string.split()))
    normalized_tf = term_reps / len_of_document

    return normalized_tf


def inverseDocumentFrequency(term, allDocs):
    num_docs_with_given_term = 0

    # Iterate through all the documents
    for doc in allDocs:
        if term.lower() in doc.lower():
            num_docs_with_given_term += 1

    if num_docs_with_given_term > 0:
        # Total number of documents
        total_num_docs = len(allDocs)

        # Calculating the IDF
        idf_val = log(1 + float(total_num_docs) / num_docs_with_given_term)
        return idf_val
    else:
        return 0


def calculate_tfidf_score(term, doc, allDocs):
    return term_frequency(term, doc) * inverseDocumentFrequency(term, allDocs)
def rank_all_pages(term):
    allDocsDict = retrieve_site_info()
    rank_dict = dict()

    for key, value in allDocsDict.items():
        score = calculate_tfidf_score(term, value, list(allDocsDict.values()))
        rank_dict[key] = score

    return rank_dict
```

```
def sentence_wise_rank_generation(sentence):
    allDocsDict = dict()
    terms = sentence.lower().split()
    for term in terms:
        if not allDocsDict:
            allDocsDict = rank_all_pages(term)
        else:
            newDict = rank_all_pages(term)
            for key in allDocsDict.keys():
                allDocsDict[key] += newDict[key]
    allDocsDict = dict(sorted(allDocsDict.items()))
    return allDocsDict


def scrape_text_data(url):

    sentences = []

    html = urllib.request.urlopen(url)
    htmlParse = BeautifulSoup(html, 'html.parser')

    for para in htmlParse.find_all('p'):
        sentences.append(' '.join(para.get_text().strip().split()))

    title = url.split('/')[-1]

    return title, ' '.join(sentences)
def retrieve_site_info():
    text_path = os.path.abspath('.').split(
        'PageRankProject')[0] + '/PageRankProject/internet/scripts/sites.txt'
    with open(text_path, 'r') as f:
        link_list = f.read().split()

    allDocs = dict()

    for url in link_list:
        key, value = scrape_text_data(url)
        allDocs[key] = value

    return allDocs


freq = sentence_wise_rank_generation("lorem ipsum")
print(freq)
```

Chapter – 5
Applications of PageRank Algorithm

The Mathematics of the PageRank algorithm is purely general and can be applied to any graph or network in any domain. There are many applications of PageRank algorithm in many fields. PageRank is regularly used in bibliometrics, social and information network analysis, and for link prediction and recommendation. It is also used for systems analysis of road networks, and in biology, chemistry, neuroscience, and physics.

5.1 PageRank Algorithm in LinkedIn Job History and the Job Migration Graph

If the pay, responsibility and the team fit are same for a finite number of companies, the analysis on LinkedIn Job History by Travis Addair can help one to decide which company will provide the most long-term value to one's career.

In the company graph data set, an edge from company A to B denotes that an employee of company A has left the company to join B.

LinkedIn publishes a list of so-called "Top Attractors", companies that job seekers actively pursue for their employment. This is the data set used.

PageRank algorithm is employed on the dataset, with a teleport probability of 0.85 and multiple edges being counted repeatedly. We can count multiple transfers from company A to B repeatedly to account for the fact that a lot of people going from A to B is a strong endorsement than a few people going from A to C.

In general, companies with a huge number of employees have higher PageRank, though there are some exceptions. There were some modifications made to this study. One modification was to add an edge from A to B for every year spent at A before transferring to B. Another modification was to add a recency bias, so that one's current position has more weight than one's previous positions.

To address the bias of company size, the study team created *Size Down-Weighted PageRank*. This applies more weights to links going to companies with few employees than companies with huge number of employees, in the year of job transfer.

A few additional steps were done to improve the previous ranking results. Another issue with the data was “skewness” towards companies with one or a handful of employees represented. To negate this, the study team removed all links in the graph to companies with fewer than 100 employees.

In conclusion, 8 of the top attractors appeared in the study’s top 10, and all the top 10 LinkedIn attractors were within the study’s top 100, after filtering companies with fewer employees than 100.

The study team believes that the Size Down- Weighted PageRank accurately mapped job transfers to company endorsements using the same intuition as the PageRank applied to web., but it also corrects for large variations in company size that would otherwise result in big companies absorbing all the PageRank.

Also, the study team concluded that it’s simply not statistically meaningful to include companies that have fewer than 100 samples in the result.

5.2 PageRank Algorithm in the most efficient way to destroy an ecosystem

Biologists have figured out the most efficient way to destroy an ecosystem based on the PageRank algorithm.

Scientists have long since known that the extinction of key species in a food web can cause collapse of the entire system, but the vast number of interactions between species makes it difficult to guess the key species. Now, computational biologists have adapted PageRank, to the problem of predicting ecological collapse, and they've created an accurate model compared to other existing models.

While several previous studies have looked at the robustness of food webs to a variety of sequences of species loss, none of them have come up with a way to identify the most devastating sequence of extinctions.

Using a modified version of PageRank, the research team was able to identify which species extinctions within a food web would lead to biggest chain-reaction of species death.

“If we can find the way of removing species so that the destruction of the ecosystem is the fastest, it means we’re ranking species by their importance," said ecologist Stefano Allesina of the University of California, Santa Barbara, who co-authored the paper published in *PLoS Computational Biology.*

Unlike previous models to the coextinction problem, the PageRank model takes into account not only the number of connections between species, but also their relative importance. *“In PageRank, you're an important website if important websites point to you. In the PageRank model, species are important if they support important species.*" i.e., Grass is important because it's eaten by gazelles, and gazelles are important because they're eaten by lions.

When the research team tested the PageRank model against existing models for predicting ecosystem collapse, they found that the new solution outperformed the old ones in each of the 12 food webs they looked at. In every case that the study team tested, the algorithm returned either the best possible solution, out of the billions of possibilities, or very close to it. In this case, the "best possible solution" is the one that predicts total ecosystem collapse using the fewest number of species extinctions.

PageRank algorithm is circular while food chains are traditionally considered to be unidirectional. For PageRank to be implemented, the researchers had to solve the problem of what to do with *dead ends: Not much eats a lion, but that doesn't necessarily mean lions aren't critical to the food chain*.

The scientists solved this problem by adding what the team calls a "*root node*," which is based on *the idea* that *all living creatures contribute to the food chain through their excrement and eventual decay*.

What the team found is that the importance of a species can be connected to the amount of matter that flows to it. *If species eat a lot of things, and a lot of things eat them, they tend to be important.*

Previous models to the problem tended to underestimate the importance of species that are lower on the food chain, Allesina said, and he hopes the new solution will encourage conservation biologists to take a broader view of species extinctions, like a more network – based view since each species is not independent. They are entangled in a network of multi-species interactions.

For ecosystems on the brink of collapse, such as marine environments taxed by overfishing, Allesina said a network-based approach to conservation could make all the difference.

5.3 PageRank in toxic waste management system

Scientists were able to use PageRank algorithm to help determine the position of water molecules in an ionic solution, enabling them to find the best, optimal ways to remove nuclear waste and toxic chemicals.

According to Aurora Clark, an associate professor at WSU, once you know the probable positions of different molecules in the solution, "… you can control the chemistry and force certain reactions to occur."

PageRank algorithm essentially maps to the likelihood of toxic chemicals to pool in the solution, enabling a waste clean-up team to quickly and effectively contain and remove the toxic or radioactive containment.

Conclusion

The PageRank algorithm reduces each page in the web to a unique number, its rank. It could be used to separate out a small set of documents which could possibly answer a large amount of queries.

To find out the rank of webpages, initially stochastic and primitivity adjustments are done to the adjacency matrix of the web graph. Then the application of power method on the resultant matrix results in a unique positive vector known as the stationary vector which gives the ranks of the webpages. The adjustments are done to ensure that the resultant matrix is a Markov matrix, and hence would converge on the application of power method.

The PageRank algorithm was truly a milestone achievement in the 20th century. Later it was found that the algorithm had far more applications outside the technical realm than we first thought. It was applied across different domains, like biology, chemistry, physics etc.

Bibliography

1. A short history of the web. CERN. https://home.cern/science/computing/birth-web/short-history-web

2. James E Pitkow. *Characterizing World Wide Web Ecologies.* PhD thesis, Georgia Institue of Technology, 1997

3. Ellen Spertus. *Parasite: Mining Structural information on the web.* Appearing in the Sixth International WWW Conference, 1997.

4. Ron Weiss. *HyPursuit: A hierarchical network search engine that exploits content-link hypertext clustering*. Appearing in the 7th ACM Conference on hypertext.

5. Lawrence Page and Sergey Brin. *The Anatomy of a Large Scale Hypertexual Web Search Engine*. Stanford University, 1995. http://infolab.stanford.edu/~backrub/google.html

6. Amy N Langville, Carl D Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.

7. Wikipedia.org. *PageRank*. https://en.wikipedia.org/wiki/PageRank

8. math.stackexchange.com. https://math.stackexchange.com/questions/2883258/prove-that-there-exists-a-vector-x-such-that-mx-x-where-m-is-a-marko

9. Stefano Allesina. *Googling Food Webs: Can an eigenvector measure species importance for co-extinctions*. University of California. Appeared in PLos Computation Biology, 2009.

10. Caleb Garling. *Researchers fight toxic waste with Google PageRank.* *https://www.wired.com/2012/02/google-pagerank-water/*. 2012.