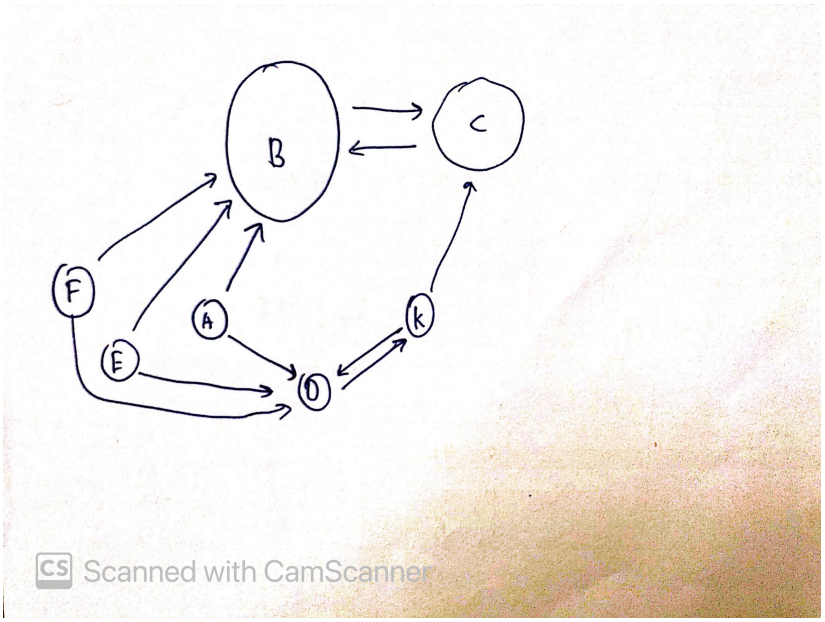
Algorithm's explanation using Markov chains

Page rank algorithms are very crucial in ensuring that information delivered to the user is as trustworthy as it can be. Page rank algorithm is based on Markov chains and how exactly they use Markov chains has been described in this report.

Page rank algorithms can be made a lot simpler by using just the number of links to a page. Higher the number of links the more popular the web-page should be. But here, a problem of spamming arises where a dummy web page with a lot of dummy links will make the algorithm useless, hence we have to introduce Markov chains to overcome this.

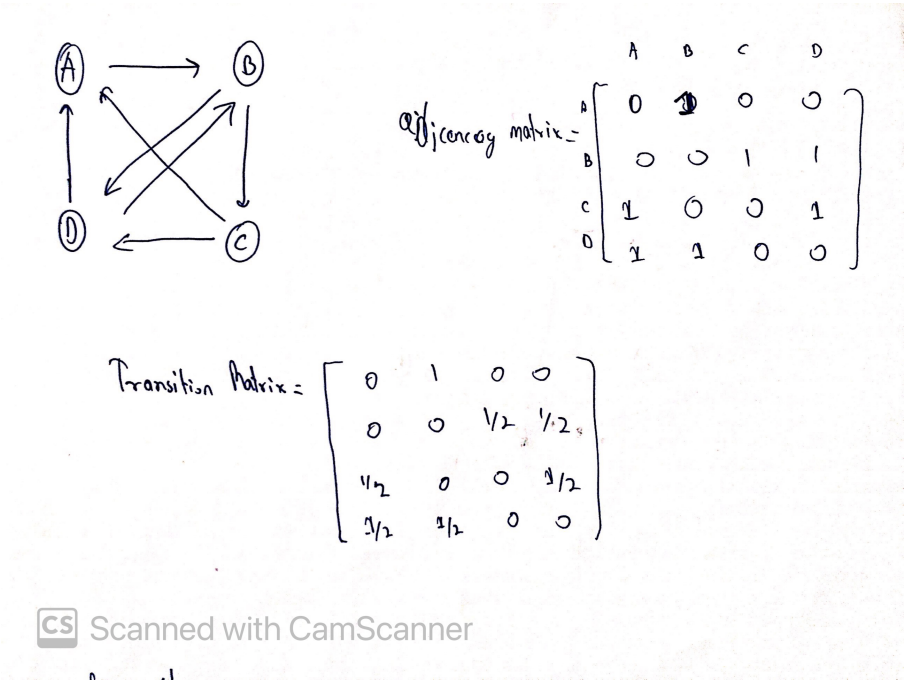
The new page rank algorithm takes into account the page rank of the page that is referencing the current page and then gives a page rank to the page. Consider the following page network for example -



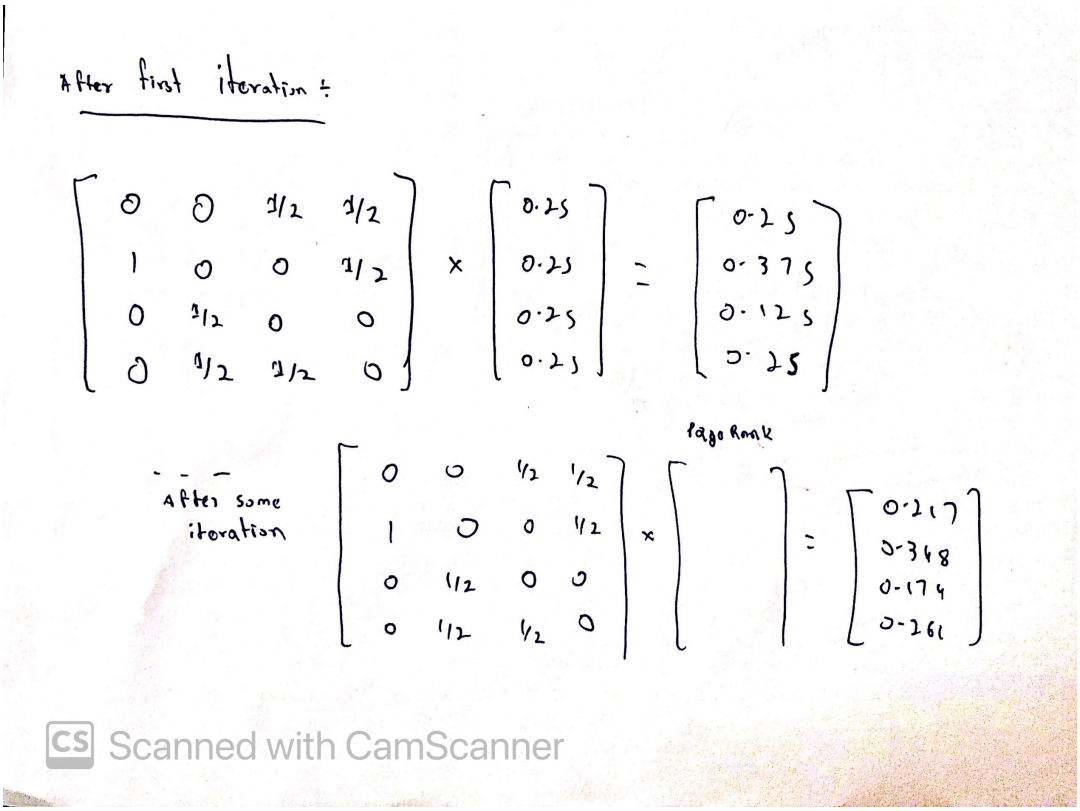
Here, the bigger circle represents higher page rank. Clearly D has more links than C and technically it must have a higher page rank than C but it doesn't .. this is because B (high page rank) references C, hence the page rank of C is increased by an even higher amount than incoming links to D.

Page rank basically gives the probability of a random user visiting the webpage.
The transition matrix of any web page network is constructed using the adjacency matrix. Each outgoing link is given an equal probability, meaning a user viewing a web page has an equal probability to go to all other web pages pointed by the current web page. Consider the following example -

The transition matrix has been built using the adjacency matrix



TO obtain the updated page ranks at every iteration, we simple multiply the transpose(transition matrix) with the page ranks.

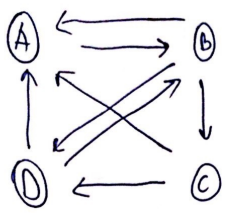


This is exactly what is happening in the following formula, it just doesn't show the transition matrix but implicitly uses it to compute the new page ranks -

$$PR(u) = \sum_{v \in in(u)} \frac{PR(v)}{|out(v)|}$$

The following scenario describes how the page rank algorithm takes into account the importance/page rank of the incoming link to a page to compute the page rank -

Scenario 2 :



A new link

D → A, ~~D → A~~ has been added to increase no. of links to A

Transition matrix :

	A	B	C	D
A	0	1	0	0
B	1/3	0	1/3	1/3
C	1/2	0	0	1/2
D	1/2	1/2	0	0

After 1st iteration :

$$\begin{bmatrix} 0 & 1/3 & 1/2 & 1/2 \\ 1 & 0 & 0 & 1/2 \\ 0 & 1/3 & 0 & 0 \\ 0 & 1/3 & 1/2 & 0 \end{bmatrix} \times \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix} = \begin{bmatrix} 0.333 \\ 0.375 \\ 0.083 \\ 0.209 \end{bmatrix}$$

after a no. of iteration :

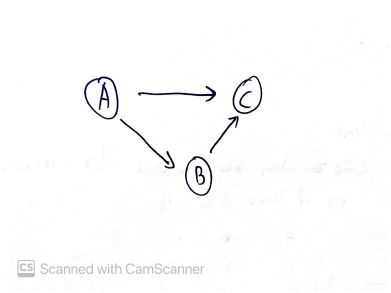
Final Page Ranks

$$\begin{bmatrix} 0.29 \\ 0.387 \\ 0.129 \\ 0.194 \end{bmatrix}$$

In the first iteration A has a higher page rank, but in the final iteration B has a much higher page rank than A. The reason for this is the fact that A having higher incoming links is referencing B, which increases its importance/page rank. Hence, the page rank also also takes into account the quality of pages referencing a page.

The problem with Edge nodes

Consider a web network like this -



Here, C is a sink. Applying the formula with the damping factor -

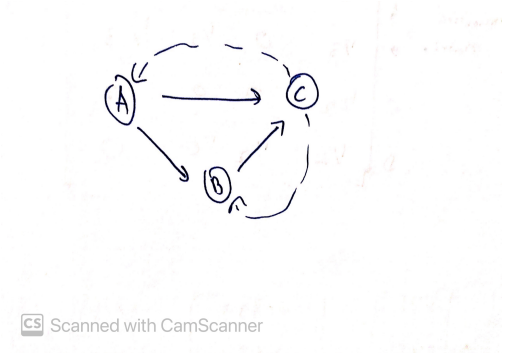
$$PR(p_i) = \frac{1-d}{N} + d \cdot \sum_{p_j \in in(p_i)} \frac{PR(p_j)}{|out(p_j)|}$$

We the get following values at starting from 1st iteration

	1	2	3	4	5	6
0.33	0.05	0.05	0.05	0.05	0.05	0.05
0.33	0.19025	0.07125	0.07125	0.07125	0.07125	0.07125
0.33	0.47075	0.2329625	0.1318125	0.1318125	0.1318125	0.1318125
Total PgRank	0.711	0.3542125	0.2530625	0.2530625	0.2530625	0.2530625

It can be seen that the page ranks don't update anymore after the 4th iteration and the sum of page ranks keeps reducing as the page ranks get accumulated at the sink.
To prevent this from happening we can add edges from this sink to all other pages and sinks in the network. This allows the web surfer to choose to got to any page if a dead end is reached.

Converting the graph to the following -



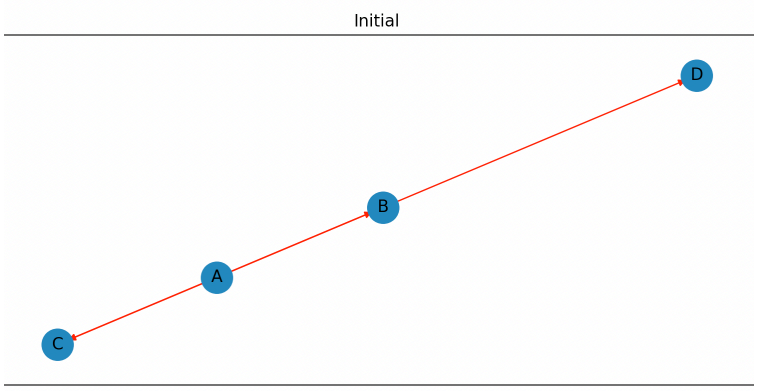
Now we can see that the page ranks add up to 1 and the issue of sinks is resolved with C having the highest page score.

	1	2	3	4	5	6	7	8
0.3333	0.1916525	0.25185694	0.22627366	0.23714963	0.23252995	0.23449553	0.23366205	0.23401788
0.3333	0.333305	0.33330925	0.33331286	0.33331593	0.33331854	0.33332076	0.33332265	0.33332425
0.3333	0.4749575	0.41476156	0.44035206	0.42948224	0.43410713	0.43214599	0.43298325	0.43263062
Total PgRank	0.999915	0.99992775	0.99993859	0.9999478	0.99995563	0.99996229	0.99996794	0.99997275

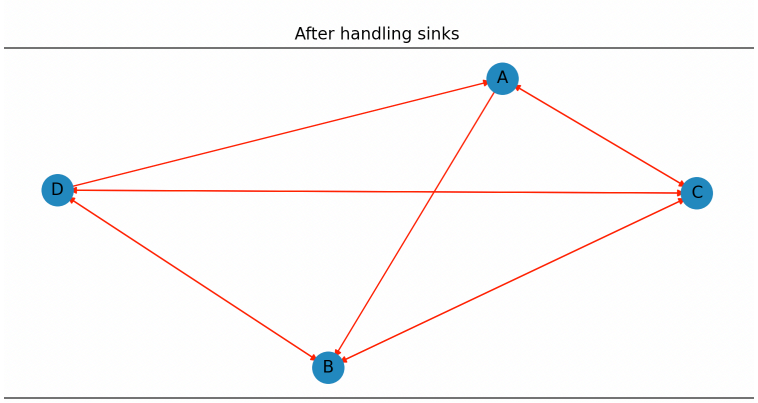
Implementation details

- I have used an adjacency matrix to determine the edges of the graph.
- handleSinks () function handles the graph in case there are any sinks. In the presence of a sink edges are drawn from the sink to all other nodes/pages/sinks in the graph
- There's a checkChange() function that checks the update rule. If the update of all page ranks from the previous page rank is **<0.01** then iterations are stopped and ranks are calculated

Results



- graphs before and after handling sinks (C and D are sinks)



Console output - the array is the

page rank of A,B,C,D respectively. The rank against each page is also printed

```
[0.19204456018518518, 0.272455150462963, 0.30143287037037036, 0.23406741898148148]
Page      Rank
C         1
B         2
D         3
A         4
```