# CS 315, Day 36 – PageRank

An interesting billboard, from several years ago, on a main highway in Silicon Valley:


Well, here are the first 70 digits of e:

2.7182818284590452353602874713526624977572470936999595749669676277240766

The answer is: 7427466391

What ***used to be*** at https://7427466391.com

A link to a job application for Google!

https://developers.google.com/search/docs/appearance/ranking-systems-guide

## Link analysis systems and PageRank

We have various systems that understand how pages link to each other as a way to determine what pages are about and which might be most helpful in response to a query. Among these is PageRank, one of our core ranking systems used when Google first launched. Those curious can learn more by reading the original PageRank research paper and patent. How PageRank works has evolved a lot since then, and it continues to be part of our core ranking systems.

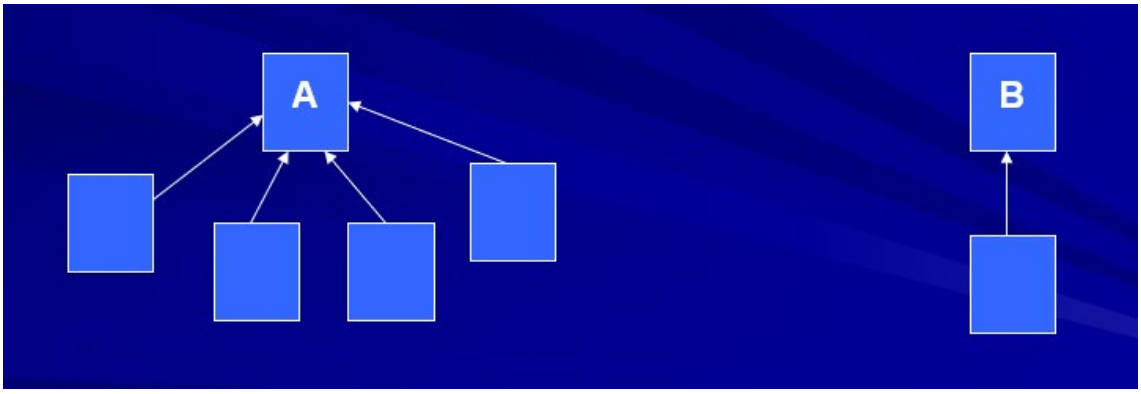
This paper with the original idea for search was written by Larry Page and Sergey Brin when they were Library Science grad students at Stanford: http://infolab.stanford.edu/~backrub/google.html

- Crawl the web and locate all* public pages
- Index the "crawled" data so it can be searched
- Rank the pages for more effective searching ( the focus of this talk )
- Each word which is searched on is linked with a list of pages (just URL's) which contain it. The pages with the highest rank are returned first.

* - impossible to get a "snapshot" of the web at a particular time.

PageRank as described in this paper just used the link structure of the World Wide Web to determine its content, ***NOT*** its content.

PageRank is NOT a simple citation index. Which do you think is more popular, $A$ or $B$ ?



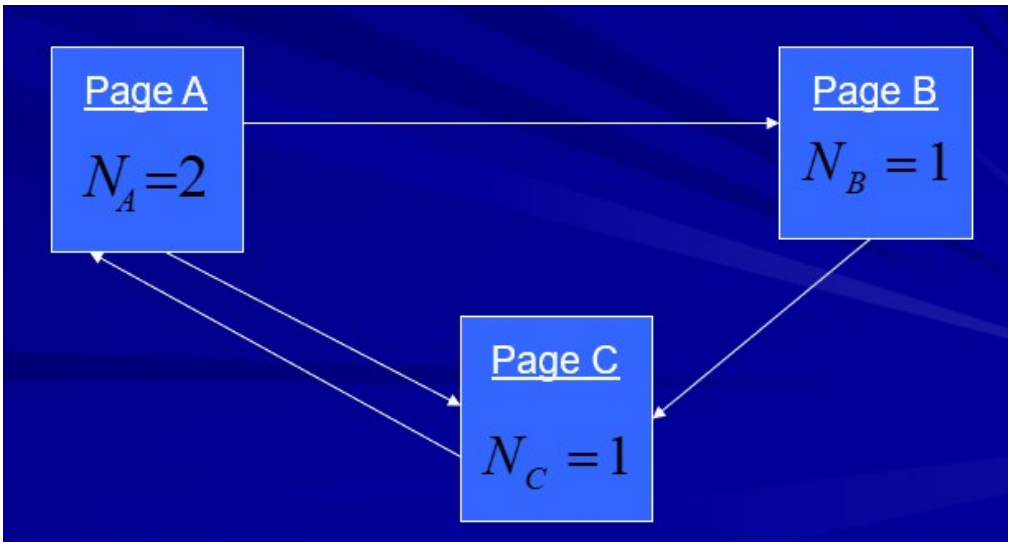

What if the links to $A$ were from unpopular or obscure pages, and the one link to $B$ was from facebook.com ?

Rankings based on simple citation indexing would be very easy to manipulate.

It's a popularity problem. Why is that person popular? Because their friends are popular. Such a circular definition.

So, in the graph of the www, each page is a vertex and each hyperlink is a directed edge.

Which of these would have the highest PageRank?




A page is popular if:

- a few very popular pages point to it, via hyperlinks
- if ***many*** not-necessarily popular pages point to it, via hyperlinks

So, how can we capture this idea mathematically?

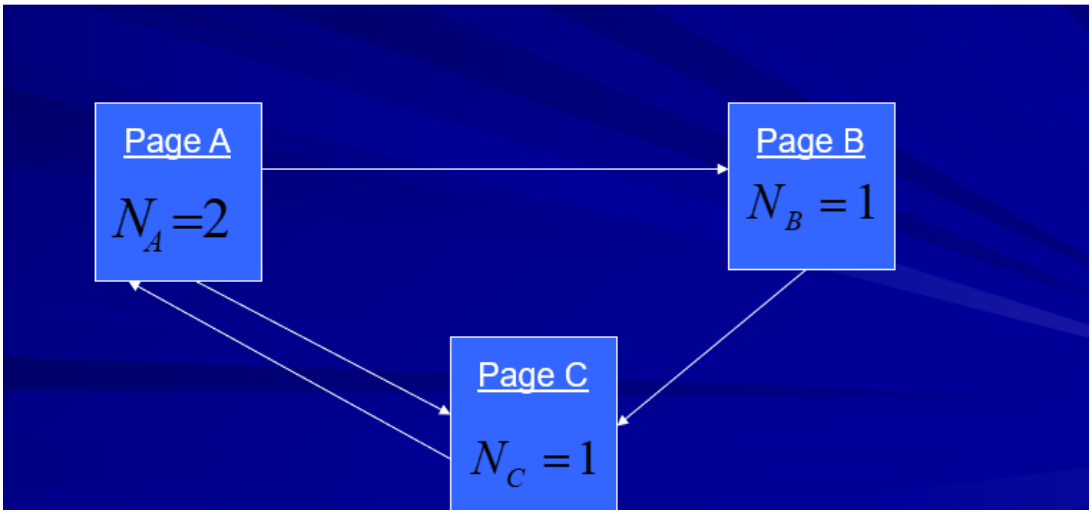
$$Rank(u) = \sum_{v \in B_u} \frac{Rank(v)}{N_v}$$

$B_u = set\ of\ pages\ with\ links\ to\ u$ $N_v = number\ of\ links\ from\ v$

Note the scaling of each page's rank.

Let's write our the equation calculating PageRank for each of the three pages:

$$
\begin{aligned}
Rank(A) &= 0 + 0 + \frac{Rank(C)}{1} \\
Rank(B) &= \frac{Rank(A)}{2} + 0 + 0 \\
Rank(C) &= \frac{Rank(A)}{2} + \frac{Rank(B)}{1} + 0
\end{aligned}
$$




Let's re-write this system of equations as a Matrix-Vector Product:

$$
\begin{pmatrix} Rank(A) \\ Rank(B) \\ Rank(C) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix} \begin{pmatrix} Rank(A) \\ Rank(B) \\ Rank(C) \end{pmatrix}
$$

And let's just switch sides:

$$
\begin{pmatrix} 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix} \begin{pmatrix} Rank(A) \\ Rank(B) \\ Rank(C) \end{pmatrix} = \begin{pmatrix} Rank(A) \\ Rank(B) \\ Rank(C) \end{pmatrix}
$$

What does this look like? You probably hit this halfway through the semester in Linear Algebra…

Here's the general equation:

$$A\vec{x} = \lambda\vec{x}$$

And if we take $\lambda = 1$, we have

$$A\vec{x} = \vec{x}$$

$\vec{x}$ is simply the ***eigenvector*** of the coefficient matrix $A$ associated with the ***eigenvalue*** $\lambda = 1$.

If the earth was a sphere of vectors, and each vector was multiplied by a rotation matrix, what vectors would be unchanged?

The vectors along the poles of rotation. These would be the ***eigenvectors***.

So, the original idea behind Google, PageRank, is using sophomore level linear algebra. That's it. Of course, it's WAAAAAY more complicated now.

Some caveats:

- Google never actually created the PageRank matrix. It would have been too big, a row and column for each web-page surfed. And it would have been extremely ***sparse***, relatively tiny amount of non-zero entries.
- The PageRank matrix is in theory column-stochastic, meaning that the sum of each column is 1, and allowing the eigenvector to be calculated iteratively.
- At some point the results of the web-crawler need to be used, so it's likely there will be dangling links, or pages that haven't been crawled yet.
- If you read Page and Brin's original paper, it's very quaint, they were definitely Library Science students and not Mathematicians.
- Also, early on Google introduced a Random Surfer term to account for bookmarks, typing in the URL directly, etc.
- The previous PageRank vector was used as a the initial guess in the next iterative calculation.

So, let's calculate the PageRank vector two ways:

1. By directly calculating the eigenvector
2. By iterating (which is what Google did/does)

Import numpy.

```
import numpy as np
```

Define the PageRank matrix $A$.

```
A = np.array([[0,0,1],
              [0.5,0,0],
              [0.5,1,0]])
```

Calculate the eigenvalues and eigenvectors of $A$.

```
eigenvalues, eigenvectors = np.linalg.eig(A)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:\n", eigenvectors)
```

Capture the eigenvector associated with $\lambda = 1$. Note that there might be some "$j's$" in the eigenvectors. In Colab, this means that there is an imaginary component to the value. But in the eigenvectors we are interested in, the coefficients of the imaginary terms are all = 0.

```
eigenvector_1 = eigenvectors[:, 0]
print(eigenvector_1)
```

We need to scale this eigenvector so all the terms sum to 1. The sum of each element in a vector is the $L_1$ norm, so we calculate this and scale (divide) each element in the eigenvector with the $L_1$ norm.

```
l1_norm = np.linalg.norm(eigenvector_1, 1)
print(l1_norm)
scaled_ev = eigenvector_1 / l1_norm
print(scaled_ev)
```

Let's check, if we multiply this scaled eigenvector with the original PageRank matrix, we should just get the same vector back as a result.

```
print(A @ scaled_ev)
```

So, this PageRank matrix is associated with the example from before. What does it tell you about which of the three pages are most popular?

***A and C are twice as popular as B***

Finally, let's calculate the PageRank eigenvector iteratively. First, define an initial vector of all 1's.

```
iter_vec=np.array([1,1,1])
print(iter_vec)
```

Then iterate through several Matrix-Vector multiplies (note that we scale the resulting vector with the $L_1$ norm before we continue. How long does it take to converge to a vector that's "close" to the original? Note, you just need to hit "Play" on the execution group. ***7 is getting "close," 10 iterations are "pretty good."***

```
iter_vec = A @ iter_vec
l1_norm = np.linalg.norm(iter_vec, 1)
iter_vec = iter_vec / l1_norm
print(iter_vec)
```

Now, based on that lab, try the following problem.

Consider the set of web-pages $\{A, B, C, D, E\}$. A forward link from one page to another can be represented as an ordered pair, so the ordered pair $(A, B)$ would indicate that page $A$ has a link pointing to page $B$.

Consider the following set of ordered pairs:
$\{ (A,C) , (B,D) , (C,A) , (E,D) , (C,E) , (D,A) , (E,A) , (C,B) , (B,E) , (E,C) , (A,B) \}$

1. Draw a directed graph of these ordered pairs on a separate piece of paper. No need to turn in, but it is helpful for you to have a picture to refer to with the web-structure.

2. Next, enter the number of forward links for each of the five pages:

$$N_A =$$
$$N_B =$$
$$N_C =$$
$$N_D =$$
$$N_E =$$

3. Create the PageRank matrix for this set of web-pages.

4. Calculate the PageRank vector.

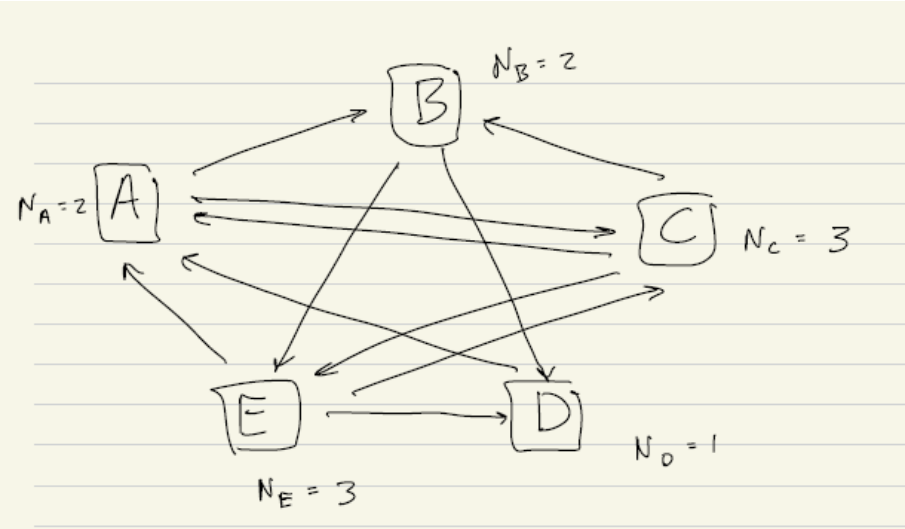
Assume that the word ***bear*** is on pages $A$, $B$, and $D$, and the word ***cat*** is on pages $C$, $D$, and $E$.

5. If the word "bear" is searched on, what would be the order of pages returned using Simple PageRank?

6. If the word "cat" was searched on, what would be the order of pages returned using Simple PageRank?

7. So if the two word phrase "bear cat" was searched on, what order of page(s) would be returned using Simple PageRank?

1. Draw a directed graph of these ordered pairs on a separate piece of paper. No need to turn in, but it is helpful for you to have a picture to refer to with the web-structure.




2. Next, enter the number of forward links for each of the five pages:

$$N_A = 2$$
$$N_B = 2$$
$$N_C = 3$$
$$N_D = 1$$
$$N_E = 3$$

3. Create the PageRank matrix for this set of web-pages.

```
A = np.array([
    [0,0,1/3,1,1/3],
    [1/2,0,1/3,0,0],
    [1/2,0,0,0,1/3],
    [0,1/2,0,0,1/3],
    [0,1/2,1/3,0,0]
])
print(A)
```

4. Calculate the PageRank vector.

```
eigenvalues, eigenvectors = np.linalg.eig(A)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:\n", eigenvectors)

eigenvector_1 = eigenvectors[:, 0]
print(eigenvector_1)

l1_norm = np.linalg.norm(eigenvector_1, 1)
print(l1_norm)
scaled_ev = eigenvector_1 / l1_norm
print(scaled_ev)

print(A @ scaled_ev)          #confirming it returns the same e.v.
```

Assume that the word ***bear*** is on pages $A$, $B$, and $D$, and the word ***cat*** is on pages $C$, $D$, and $E$.

5. If the word "bear" is searched on, what would be the order of pages returned using Simple PageRank?

$$A = 0.2778$$
$$B = 0.2037$$
$$D = 0.1574$$

6. If the word "cat" was searched on, what would be the order of pages returned using Simple PageRank?

$$C = 0.1944$$
$$E = 0.1667$$
$$D = 0.1574$$

7. So if the two word phrase "bear cat" was searched on, what order of page(s) would be returned using Simple PageRank?
   ***D, since it is the only web page with both bear and cat. Then everything else would be in PageRank order, A, B, C, E***