Markov Chains, Random Walks, and PageRank

Katrina Florendo and Keerti Mukkamala

December 2020

1 Theory Behind PageRank

PageRank was one of the first algorithms used by Google Search to determine the rank of web pages retrieved by search engine results based on relative importance. This link analysis algorithm was initially developed at Stanford University by Larry Page and Sergey Brin, in 1996 as a part of a research project, with the first prototype of the Google Search engine being released in 1998.

We begin with a set of N webpages, and we can crawl the web to determine the structure of link relations between these pages. In this algorithm, the web is modeled as an oriented graph, where each of the N vertices represent one of the N pages of the web and the edges represent the links between these pages. A given website may consist of multitudes of pages, as each page corresponds to a unique url.

Each page is assigned an initial rank of $1/N$. The rank of each page is iteratively updated by taking the sum of the weights of each page that links to it and dividing it by the number of outgoing links from a given page. Pages with no outward links equally redistribute their rank among the other pages of the graph, and this redistrubtion is applied to each page of the graph until the ranks of the pages stabilize. To model the fact that users stop searching, the algorithm makes use of a damping factor at each iteration. Google uses a factor of 0.15.

The oriented graph can be summarized by a Markov Transition matrix P, where each column represents a launching or departing page and each row represents an arrival page.Some critical properties of Markov transitions matrices are that they always have an eigenvalue of 1 (the Perron-Frobenius eigenvalue of this matrix) and there exists a corresponding eigenvector (with an eigenvalue of 1) such that each component is between 0 and 1 inclusive, with the sum of the components being 1.

In this algorithm, we consider X_n, a variable that takes on a value in the set of N pages and represents the page we are at after an n-step random walk. The entries of matrix P at the i-th row and j-th column or P_{i_j} represent the conditional probability that we are at the i-th arrival page on the $n+1$ step, given that we are at the j-th departing page at the nth step:

$$P_{i_j} = \text{prob}(X_{n+1}{=}i | X_n{=}j)$$

The probabilities of the 2nd step can be easily encapsulated by the matrix P^2, the 3rd step by P^3, and so on. This can be generalized to the m-th step as:

$$P_{i_j}^m = \text{prob}(X_{n+m}{=}i|X_n{=}j)$$

At sufficiently large values of m, we find stabilization, and that all the columns of P^m are equal, meaning that after a sufficiently large number of steps, the likelihood of hitting each of the N pages is independent from the starting page.

Let us now study the eigenvector π, the vector such that $P\pi{=}\lambda\pi$ where $\lambda{=}1$. The i-th component of the vector represents the probability of being on page i at the n step, which makes π the probability distribution at this step. This is also equivalent to the probability distribution at the n+1 step, as the vector π represents the stationary distribution. This probabilities in this vector allow us to order the N pages in order of relative importance, where the page with the highest corresponding probability is determined to be most important.

Although the PageRank algorithm in its original form is not widely used today, it has paved the wave for other web search and ranking algorithms to be used today.

2 Julia Code

The PageRank algorithm was implemented in Julia using Jupyter Lab. It uses 3 packages—LinearAlgebra, Base, and CSV—to perform matrix and vector calculations as well as parse input data.

2.1 Input Data

In a webgraph, pages are represented as nodes and the links between pages are represented as directed edges.

The format of the input data is an array of subarrays. The length of the main array is the number of nodes in the webgraph, and each index of the main array corresponds to the ID of the node. The subarray at the corresponding index contains all the nodes that the starting node is connected to (an empty subarray denotes a nodes that does not link to any other nodes).

Two small test datasets were created for a proof of concept.

```
data1 = [[2], [1, 3], [1, 2, 5], [1], [2, 3, 4]]
data2 = [[2], [1, 3], [1, 2, 5], [], [2, 3, 4, 1]]
```

An if-else block is used to quickly switch between data sets. In this case, we choose dataset 1. The variable `D` stores the chosen dataset. After choosing a dataset, the variable `numPages` stores the number of nodes in the webgraph.

```
dataset = 1;
if dataset == 1
    D = data1;
elseif dataset == 2
    D = data2;
elseif dataset == 3
    D = chameleonData;
end
numPages = size(D)[1];
```

2.2 Creating the Markov Matrix (P)

A matrix `P` is initialized with `numPages` rows and `numPages` columns with 0 for each entry. A for loop is used to iterate over every node and create a Markov matrix. The entry in the i-th row and j-th column corresponds to the probability of starting at node j and going to node i at the next step in a random walk.

For pages that do not link to any other pages (i.e. have an empty list at their corresponding index), each entry in the column is set to `1/numPages` to randomly restart the random walk if a dead end is reached.

```
P = zeros(Float32, numPages, numPages);
for i = 1:numPages
    cur = D[i];
    numLinks = size(cur)[1];
    if numLinks == 0
    # if page contains no links, randomly restart walk
        for j = 1:numPages
            setindex!(P, 1/numPages, j, i);
        end
    else
        for j = 1:numLinks
            setindex!(P, 1/numLinks, cur[j], i);
        end
    end
end
```

2.3 Applying the Damping Factor

The following code applies a damping factor of $\beta = 0.15$ to `P`.

```
beta = 0.15;
onesMatrix = fill(1.0, (numPages, numPages));
Q = (1/numPages) * onesMatrix;
P = (1 - beta)*P + beta*Q;
```

2.4 Computing the Steady State Vector

The `eigen()` function is used to compute the eigenvectors and eigenvalues of `P`. The eigenvector with corresponding eigenvalue 1 is selected and set to the variable `vec1` (Note: this is the vector at the last index of the array of all returned eigenvectors). The components of the vector are scaled so that the sum of the components equals 1, and thus each component corresponds to the probability of being at that node during a random walk.

```
valP, vecP = eigen(P)
vec1 = vecP[:, size(D)[1]];
sumVec = 0;
for i = 1:numPages
    sumVec = sumVec + vec1[i];
end
scaleFactor = 1 / sumVec;
steadyStateVec = scaleFactor * vec1;
```

2.5 Output List of Rankings

A matrix `R` is created with `numPages` rows and 2 columns. The first column contains the ID of a node and the second column contains the corresponding probability. `R` is then sorted by probability.

Finally, the array `ranks` is populated with the IDs of the nodes from highest rank to lowest rank.

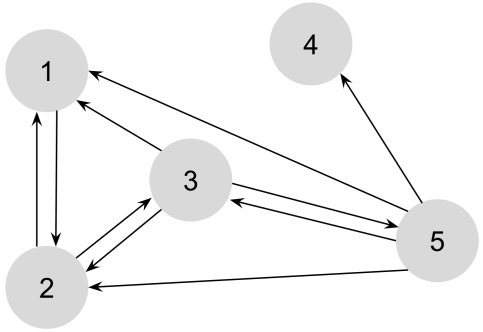
```
R = zeros(numPages, 2);
for i = 1:numPages
    setindex!(R, i, i, 1);
    setindex!(R, convert(Float64, steadyStateVec[i
    ]), i, 2);
end
R = R[sortperm(R[:, 2]), :]; # sort by the second
    column

ranks = Array{Int32}(undef, numPages);
for i = 1:numPages
    ranks[numPages + 1 - i] = R[i][1];
end
```

3 Results

We will examine the ranking results for dataset 2, also shown in Figure 1. The results for the intermediate steps of the algorithm can be seen in the attached document "PageRank Algorithm on Data Set 2".

```
data2 = [[2], [1, 3], [1, 2, 5], [], [2, 3, 4, 1]]
```

Figure 1: Visualization of dataset 2 webgraph



The output ranks array is as follows:

```
ranks = [2, 1, 3, 5, 4]
```

The rankings seem appropriate for the webgraph. Pages 2 and 3 rank highest since they both have three incoming links. Page 3 ranks third highest with two incoming links and page 5 ranks fourth highest with one incoming link. Page 4 is somewhat isolated from the rest of the graph, so it is appropriate that it is ranked last.

4 Project Extension: Real World Data

As an extension to the PageRank project, we applied the algorithm to real-world data. The code was run on a dataset from Stanford University's SNAP research group representing a page-page network on chameleon-related articles. The graph has 2,277 nodes and 31,421 edges [1].

The original dataset is stored in a .csv file. We used Julia's CSV package to parse the data and put it in the input data format as described in section 2.1. The following code performs this procedure.

```
chameleonNumNodes = 2277;
chameleonData = [[] for i=1:chameleonNumNodes];

f = CSV.File("musae_chameleon_edges.csv");
count = 1;
for row in f
    startNode = row.id1 + 1;
    finishNode = row.id2 + 1;
    if chameleonData[startNode] == undef
        chameleonData[startNode] = [];
    end
    append!(chameleonData[startNode], [finishNode
    ]);
    count = count + 1
end
```

The results for the intermediate steps of the algorithm and the final ranks array can be seen in the attached document "PageRank Algorithm on Chameleon Data Set". Despite the larger size of the webgraph, the algorithm ran rather quickly.

5 Analysis of PageRank Successors

While PageRank was the first algorithm used by Google to determine the order of their search results, all patents for the Page Rank algorithm expired in 2019 and were not renewed, and the original form of the algorithm has not been used since 2006. Given the rapid growth of the web and the sheer size and heterogeneity of the data present online, there is an increased need for flexibility in the ranking, as each user has their own definition of the relative importance of given input or query. Although accounting for this in a personalized form of PageRank may seem feasible in theory, the implementation is very resource-intensive and inefficient.

Around the same time when PageRank was developed, Professor Jon Kleinberg from Cornell University developed another solution to rank queries from a web search: the HITS (Hyperlink-Induced Topic Search) algorithm. Kleinberg sought to include more of a human perspective, noting that a search process should be more than simply comparing query words against documents to find matches. Webpages that include the query words are not necessarily relevant.

The HITS algorithm identifies pages that are authorities (pages that are official brand/government/producer sites and contain valuable information) and hubs (pages that link to and promote authorities). HITS identifies authorities and hubs by assigning each page two recursively defined values: an authority weight and a hub weight. A high authority weight means that several "hub" sites link to a site, and a high hub weight means that it links to several authority sites.

Much like its predecessor, HITS uses the link structure of a webgraph to rank and order pages by relative importance and value. The key difference is that HITS is used on a small subgraph known as the seed, and this subgraph is determined based on the user inputted query. HITS ranks each node of the seed using the authority/hub weighting scheme discussed earlier.

6 Takeaways

Through this study, we learned how to apply our knowledge of Markov matrices and random walks to understand and implement the PageRank algorithm. We wrote code in Julia that successfully ranks a list of websites. We took the project a step further by not only applying the Julia code to a real-world dataset, but also by exploring the successors of PageRank, including the HITS algorithm. Ultimately, we saw how linear algebra could be used as a powerful tool behind the things we encounter everyday, like search engines.

References

[1] Wikipedia Article Networks, Stanford SNAP, https://snap.stanford.edu/data/wikipedia-article-networks.html.

[2] STAT 455 Lecture Notes, Queens University Canada, mast.queensu.ca/ stat455/lecturenotes/set3.pdf.

[3] "The PageRank Computation." Stanford NLP, nlp.stanford.edu/IR-book/html/htmledition/the-pagerank-computation-1.html.

[4] Garcia-Molina, Hector. The Stanford Pagerank Project, pagerank.stanford.edu/.

[5] Schwartz, Barry. "Former Google Engineer: Google Hasn't Used PageRank Since 2006." Seroundtable.com, Search Engine Roundtable, 16 July 2019, www.seroundtable.com/google-hasnt-used-pagerank-since-2006-27891.html.

[6] Rousseau, Christiane. D´Epartement De Math´Ematiques Et De Statistique and CRM, Universit´e De Montr´Eal , dmuw.zum.de/images/f/f8/Google_klein_2.pdf.

PageRank Algorithm on Data Set 2

Katrina Florendo and Keerti Mukkamala | 21241 Matrices and Linear Transformations | Professor Offner

In [35]:

```
using Pkg
# Pkg.add("LinearAlgebra")
# Pkg.add("Base")
# Pkg.add("CSV")
```

In [36]:

```
using LinearAlgebra
using Base
using CSV
```

Small Data Sets

In [37]:

```
data1 = [[2], [1, 3], [1, 2, 5], [1], [2, 3, 4]]
```

Out[37]:

```
5-element Array{Array{Int64,1},1}:
 [2]
 [1, 3]
 [1, 2, 5]
 [1]
 [2, 3, 4]
```

In [38]:

```
data2 = [[2], [1, 3], [1, 2, 5], [], [2, 3, 4, 1]]
```

Out[38]:

```
5-element Array{Array{Any,1},1}:
 [2]
 [1, 3]
 [1, 2, 5]
 []
 [2, 3, 4, 1]
```

Large Data Set

In [39]:

```
chameleonNumNodes = 2277;
chameleonData = [[] for i=1:chameleonNumNodes];
```

In [40]:

```
f = CSV.File("musae_chameleon_edges.csv");
count = 1;
for row in f
    startNode = row.id1 + 1;
    finishNode = row.id2 + 1;
    if chameleonData[startNode] == undef
        chameleonData[startNode] = [];
    end
    append!(chameleonData[startNode], [finishNode]);
    count = count + 1
end
```

In [41]:

```
chameleonData
```

Out[41]:

```
2277-element Array{Array{Any,1},1}:
 [1668, 2131, 1162, 2157, 1992]
 [1906, 1844, 2227, 1848, 1721, 1912, 1180, 1662, 2139, 1760, 556, 51,
1925, 1226, 1804, 2058, 1902]
 [2031, 1238, 922, 2192, 352, 1715, 1901, 1426, 1341, 2091, 2146, 2178,
1289, 2276, 2210, 882, 2154, 2187, 1521, 221]
 [266]
 [1557, 1940, 901, 1865]
 [1844, 2227, 1848, 1721, 1912, 556, 2085, 1902, 821]
 [2264, 1977, 1742, 2267]
 [1463, 2196, 1990]
 [2032, 1940, 1910, 2264, 1977, 1742]
 [1334, 1940, 1498, 1055, 2264, 1977, 2236, 807, 1357, 1742]
 [761, 1399, 2166, 1531, 1024, 898, 1410, 1919, 1922, 1286  …  374, 124
8, 1888, 2014, 367, 1128, 1896, 371, 1899, 1014]
 [1325]
 [1463, 2264, 1566, 651, 337, 1990, 1357, 123]
 ⋮
 [2247, 1940, 1753]
 [2247, 2264, 1977, 1742, 2240]
 [1499, 1940]
 [1428, 1018, 1940, 1719, 2198, 1694, 1981, 1857, 1665, 2241, 2273, 126
6]
 [2111, 1152, 1415]
 [2257, 1940, 2261, 2264, 1977, 1742]
 [1332, 1693, 2079]
 [1428, 1018, 1719, 2198, 1694, 1981, 2269, 1857, 1920, 2241, 1415, 166
5, 1931, 1266]
 [2055, 1914, 2141, 663, 154]
 [776]
 [1715, 2154, 2141, 2175, 1924, 1163, 2184, 1962, 2027, 881]
 [739, 521, 1918, 2048, 1861, 1800, 2213, 2277, 2247, 1357, 2158]
```

Choose Data Set

Choose a data set by setting the dataset variable equal to 1 or 2.

In [42]:

```
dataset = 2;
```

In [43]:

```
if dataset == 1
    D = data1;
elseif dataset == 2
    D = data2;
elseif dataset == 3
    D = chameleonData;
end
```

Out[43]:

```
5-element Array{Array{Any,1},1}:
 [2]
 [1, 3]
 [1, 2, 5]
 []
 [2, 3, 4, 1]
```

In [44]:

```
numPages = size(D)[1]
```

Out[44]:

```
5
```

Convert data set into Markov matrix (P)

In [45]:

```
P = zeros(Float32, numPages, numPages);
```

In [46]:

```
for i = 1:numPages
    cur = D[i];
    numLinks = size(cur)[1];
    if numLinks == 0 # if page contains no links, randomly restart walk
        for j = 1:numPages
            setindex!(P, 1/numPages, j, i);
        end
    else
        for j = 1:numLinks
            setindex!(P, 1/numLinks, cur[j], i);
        end
    end
end
```

In [47]:

```
P
```

Out[47]:

```
5×5 Array{Float32,2}:
 0.0  0.5  0.333333  0.2  0.25
 1.0  0.0  0.333333  0.2  0.25
 0.0  0.5  0.0       0.2  0.25
 0.0  0.0  0.0       0.2  0.25
 0.0  0.0  0.333333  0.2  0.0
```

Apply damping factor to P

In [48]:

```
beta = 0.15;
onesMatrix = fill(1.0, (numPages, numPages))
Q = (1/numPages) * onesMatrix
```

Out[48]:

```
5×5 Array{Float64,2}:
 0.2  0.2  0.2  0.2  0.2
 0.2  0.2  0.2  0.2  0.2
 0.2  0.2  0.2  0.2  0.2
 0.2  0.2  0.2  0.2  0.2
 0.2  0.2  0.2  0.2  0.2
```

In [49]:

```
P = (1 - beta)*P + beta*Q
```

Out[49]:

```
5×5 Array{Float64,2}:
 0.03  0.455  0.313333  0.2  0.2425
 0.88  0.03   0.313333  0.2  0.2425
 0.03  0.455  0.03      0.2  0.2425
 0.03  0.03   0.03      0.2  0.2425
 0.03  0.03   0.313333  0.2  0.03
```

Determine the eigenvalues and eigenvectors of P

In [50]:

```
valP, vecP = eigen(P)
```

Out[50]:

```
Eigen{Float64,Float64,Array{Float64,2},Array{Float64,1}}
values:
5-element Array{Float64,1}:
 -0.556523738666121
 -0.2706645610490177
 -0.1115947138112296
  0.25878300990374925
  1.0000000061558172
vectors:
5×5 Array{Float64,2}:
 -0.294413   -0.0239725  -0.0710857   0.324982  -0.535976
  0.656939   -0.0899865  -0.167483    0.526972  -0.695829
 -0.599756    0.512165    0.046191    0.155132  -0.417644
 -0.0980718   0.370867   -0.591655   -0.710315  -0.121917
  0.335301   -0.769073    0.784033   -0.296771  -0.198144
```

In [51]:

```
vec1 = vecP[:, size(D)[1]]
```

Out[51]:

```
5-element Array{Float64,1}:
 -0.5359764771432189
 -0.6958291099866194
 -0.41764400598317114
 -0.12191663605930171
 -0.19814359379673693
```

In [52]:

```
P^100000
```

Out[52]:

```
5×5 Array{Float64,2}:
 0.272305   0.272305   0.272305   0.272305   0.272305
 0.353518   0.353518   0.353518   0.353518   0.353518
 0.212185   0.212185   0.212185   0.212185   0.212185
 0.0619401  0.0619401  0.0619401  0.0619401  0.0619401
 0.100667   0.100667   0.100667   0.100667   0.100667
```

Scale the eigenvector corresponding to eigenvalue 1

We scale the eigenvector corresponding to eigenvalue 1 so that the sum of the components = 1

In [53]:

```
sumVec = 0
for i = 1:numPages
    sumVec = sumVec + vec1[i];
end
scaleFactor = 1 / sumVec
```

Out[53]:

```
-0.507740549621882
```

In [54]:

```
steadyStateVec = scaleFactor * vec1
```

Out[54]:

```
5-element Array{Float64,1}:
 0.27213699108909806
 0.3533006547475111
 0.21205479714417988
 0.0619020198008008
 0.10060553721841013
```

Create list of rankings

Create an n by 2 matrix R (n is the number of pages) where R_i1 is the page index and R_i2 is its corresponding probability.

In [55]:

```
R = zeros(numPages, 2)
```

Out[55]:

```
5×2 Array{Float64,2}:
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
```

In [56]:

```
for i = 1:numPages
    setindex!(R, i, i, 1);
    setindex!(R, convert(Float64, steadyStateVec[i]), i, 2);
end
R
```

Out[56]:

```
5×2 Array{Float64,2}:
 1.0  0.272137
 2.0  0.353301
 3.0  0.212055
 4.0  0.061902
 5.0  0.100606
```

In [57]:

```
R = R[sortperm(R[:, 2]), :] # sort by the second column
```

Out[57]:

```
5×2 Array{Float64,2}:
 4.0  0.061902
 5.0  0.100606
 3.0  0.212055
 1.0  0.272137
 2.0  0.353301
```

In [58]:

```
ranks = Array{Int32}(undef, numPages);
for i = 1:numPages
    ranks[numPages + 1 - i] = R[i][1];
end
ranks
```

Out[58]:

```
5-element Array{Int32,1}:
 2
 1
 3
 5
 4
```

PageRank Algorithm on Chameleon Data Set

Katrina Florendo and Keerti Mukkamala | 21241 Matrices and Linear Transformations | Professor Offner

In [76]:

```
using Pkg
# Pkg.add("LinearAlgebra")
# Pkg.add("Base")
# Pkg.add("CSV")
```

In [77]:

```
using LinearAlgebra
using Base
using CSV
```

Small Data Sets

In [78]:

```
data1 = [[2], [1, 3], [1, 2, 5], [1], [2, 3, 4]]
```

Out[78]:

```
5-element Array{Array{Int64,1},1}:
 [2]
 [1, 3]
 [1, 2, 5]
 [1]
 [2, 3, 4]
```

In [79]:

```
data2 = [[2], [1, 3], [1, 2, 5], [], [2, 3, 4, 1]]
```

Out[79]:

```
5-element Array{Array{Any,1},1}:
 [2]
 [1, 3]
 [1, 2, 5]
 []
 [2, 3, 4, 1]
```

Large Data Set

In [80]:

```
chameleonNumNodes = 2277;
chameleonData = [[] for i=1:chameleonNumNodes];
```

In [81]:

```
f = CSV.File("musae_chameleon_edges.csv");
count = 1;
for row in f
    startNode = row.id1 + 1;
    finishNode = row.id2 + 1;
    if chameleonData[startNode] == undef
        chameleonData[startNode] = [];
    end
    append!(chameleonData[startNode], [finishNode]);
    count = count + 1
end
```

In [82]:

```
chameleonData
```

Out[82]:

```
2277-element Array{Array{Any,1},1}:
 [1668, 2131, 1162, 2157, 1992]
 [1906, 1844, 2227, 1848, 1721, 1912, 1180, 1662, 2139, 1760, 556, 51,
1925, 1226, 1804, 2058, 1902]
 [2031, 1238, 922, 2192, 352, 1715, 1901, 1426, 1341, 2091, 2146, 2178,
1289, 2276, 2210, 882, 2154, 2187, 1521, 221]
 [266]
 [1557, 1940, 901, 1865]
 [1844, 2227, 1848, 1721, 1912, 556, 2085, 1902, 821]
 [2264, 1977, 1742, 2267]
 [1463, 2196, 1990]
 [2032, 1940, 1910, 2264, 1977, 1742]
 [1334, 1940, 1498, 1055, 2264, 1977, 2236, 807, 1357, 1742]
 [761, 1399, 2166, 1531, 1024, 898, 1410, 1919, 1922, 1286  …  374, 124
8, 1888, 2014, 367, 1128, 1896, 371, 1899, 1014]
 [1325]
 [1463, 2264, 1566, 651, 337, 1990, 1357, 123]
 ⋮
 [2247, 1940, 1753]
 [2247, 2264, 1977, 1742, 2240]
 [1499, 1940]
 [1428, 1018, 1940, 1719, 2198, 1694, 1981, 1857, 1665, 2241, 2273, 126
6]
 [2111, 1152, 1415]
 [2257, 1940, 2261, 2264, 1977, 1742]
 [1332, 1693, 2079]
 [1428, 1018, 1719, 2198, 1694, 1981, 2269, 1857, 1920, 2241, 1415, 166
5, 1931, 1266]
 [2055, 1914, 2141, 663, 154]
 [776]
 [1715, 2154, 2141, 2175, 1924, 1163, 2184, 1962, 2027, 881]
 [739, 521, 1918, 2048, 1861, 1800, 2213, 2277, 2247, 1357, 2158]
```

Choose Data Set

Choose a data set by setting the dataset variable equal to 1 or 2.

In [83]:

```
dataset = 3;
```

In [84]:

```
if dataset == 1
    D = data1;
elseif dataset == 2
    D = data2;
elseif dataset == 3
    D = chameleonData;
end
```

Out[84]:

```
2277-element Array{Array{Any,1},1}:
 [1668, 2131, 1162, 2157, 1992]
 [1906, 1844, 2227, 1848, 1721, 1912, 1180, 1662, 2139, 1760, 556, 51,
1925, 1226, 1804, 2058, 1902]
 [2031, 1238, 922, 2192, 352, 1715, 1901, 1426, 1341, 2091, 2146, 2178,
1289, 2276, 2210, 882, 2154, 2187, 1521, 221]
 [266]
 [1557, 1940, 901, 1865]
 [1844, 2227, 1848, 1721, 1912, 556, 2085, 1902, 821]
 [2264, 1977, 1742, 2267]
 [1463, 2196, 1990]
 [2032, 1940, 1910, 2264, 1977, 1742]
 [1334, 1940, 1498, 1055, 2264, 1977, 2236, 807, 1357, 1742]
 [761, 1399, 2166, 1531, 1024, 898, 1410, 1919, 1922, 1286  …  374, 124
8, 1888, 2014, 367, 1128, 1896, 371, 1899, 1014]
 [1325]
 [1463, 2264, 1566, 651, 337, 1990, 1357, 123]
 ⋮
 [2247, 1940, 1753]
 [2247, 2264, 1977, 1742, 2240]
 [1499, 1940]
 [1428, 1018, 1940, 1719, 2198, 1694, 1981, 1857, 1665, 2241, 2273, 126
6]
 [2111, 1152, 1415]
 [2257, 1940, 2261, 2264, 1977, 1742]
 [1332, 1693, 2079]
 [1428, 1018, 1719, 2198, 1694, 1981, 2269, 1857, 1920, 2241, 1415, 166
5, 1931, 1266]
 [2055, 1914, 2141, 663, 154]
 [776]
 [1715, 2154, 2141, 2175, 1924, 1163, 2184, 1962, 2027, 881]
 [739, 521, 1918, 2048, 1861, 1800, 2213, 2277, 2247, 1357, 2158]
```

In [85]:

```
numPages = size(D)[1]
```

Out[85]:

```
2277
```

Convert data set into Markov matrix (P)

In [86]:

```
P = zeros(Float32, numPages, numPages);
```

In [87]:

```
for i = 1:numPages
    cur = D[i];
    numLinks = size(cur)[1];
    if numLinks == 0 # if page contains no links, randomly restart walk
        for j = 1:numPages
            setindex!(P, 1/numPages, j, i);
        end
    else
        for j = 1:numLinks
            setindex!(P, 1/numLinks, cur[j], i);
        end
    end
end
```

In [88]:

```
P
```

Out[88]:

```
2277×2277 Array{Float32,2}:
 0.0  0.0  0.0   0.0  0.0  0.0  0.0   …  0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0   …  0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0   …  0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 ⋮                        ⋮          ⋱                       ⋮
 0.0  0.0  0.0   0.0  0.0  0.0  0.0   …  0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.25     0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0714286  0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0   …  0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.05  0.0  0.0  0.0  0.0   …  0.0        0.0  0.0  0.0  0.0
 0.0  0.0  0.0   0.0  0.0  0.0  0.0      0.0        0.0  0.0  0.0  0.09
09091
```

Apply damping factor to P

In [89]:

```
beta = 0.15;
onesMatrix = fill(1.0, (numPages, numPages))
Q = (1/numPages) * onesMatrix
```

Out[89]:

```
2277×2277 Array{Float64,2}:
 0.000439174  0.000439174  0.000439174  …  0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174  …  0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174  …  0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 ⋮                                    ⋱   ⋮
 0.000439174  0.000439174  0.000439174  …  0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174  …  0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174  …  0.000439174  0.000439174
 0.000439174  0.000439174  0.000439174     0.000439174  0.000439174
```

In [90]:

```
P = (1 - beta)*P + beta*Q
```

Out[90]:

```
2277×2277 Array{Float64,2}:
 6.58762e-5  6.58762e-5  6.58762e-5  …  6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5  …  6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5  …  6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
⋮                                    ⋱              ⋮
 6.58762e-5  6.58762e-5  6.58762e-5  …  6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5  …  6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  0.0425659   …  6.58762e-5  6.58762e-5  6.58762
e-5
 6.58762e-5  6.58762e-5  6.58762e-5     6.58762e-5  6.58762e-5  0.07733
86
```

Determine the eigenvalues and eigenvectors of P

In [91]:

```
valP, vecP = eigen(P)
```

Out[91]:

```
Eigen{Complex{Float64},Complex{Float64},Array{Complex{Float64},2},Array
{Complex{Float64},1}}
values:
2277-element Array{Complex{Float64},1}:
 -0.8500000000000029 + 0.0im
 -0.8500000000000028 + 0.0im
 -0.8500000000000019 + 0.0im
 -0.8499999999999979 + 0.0im
 -0.7361297104549464 + 0.0im
 -0.7177296647120031 + 0.0im
 -0.6676758469284158 + 0.0im
 -0.6579389172442733 + 0.0im
 -0.6373860837196031 + 0.0im
 -0.6194065307840355 + 0.0im
 -0.6126859528691349 + 0.0im
 -0.6010422728990364 + 0.0im
 -0.5914298321808347 + 0.0im
                     ⋮
  0.8455699295688843 + 0.0im
  0.8474783924428805 + 0.0im
  0.8481267611869701 + 0.0im
  0.8499999999999935 + 0.0im
  0.8499999999999962 + 0.0im
  0.8499999999999971 + 0.0im
  0.8500000000000036 + 0.0im
  0.8500000000000039 + 0.0im
  0.8500000000000069 + 0.0im
  0.8500000010267409 + 0.0im
  0.8500000092506524 + 0.0im
  1.0000000135940077 + 0.0im
vectors:
2277×2277 Array{Complex{Float64},2}:
 -1.64586e-17+0.0im    5.0429e-19+0.0im  …  -0.000840021+0.0im
 -3.16951e-17+0.0im   -1.7783e-16+0.0im       -0.0143231+0.0im
  1.02804e-17+0.0im  -3.43391e-18+0.0im      -0.000840021+0.0im
 -6.19374e-19+0.0im  -8.57553e-19+0.0im      -0.000840021+0.0im
 -7.02271e-18+0.0im  -2.84931e-18+0.0im      -0.000840021+0.0im
  5.93584e-17+0.0im   9.28946e-17+0.0im  …   -0.00243825+0.0im
 -1.10263e-17+0.0im   8.15445e-18+0.0im      -0.000840021+0.0im
 -1.33397e-17+0.0im  -1.88896e-17+0.0im      -0.000840021+0.0im
 -1.63478e-16+0.0im  -1.95245e-16+0.0im       -0.00571248+0.0im
  -1.0985e-17+0.0im  -4.29736e-17+0.0im      -0.000840021+0.0im
  3.11436e-17+0.0im  -6.99497e-17+0.0im  …   -0.000840021+0.0im
   1.9442e-17+0.0im   4.43043e-17+0.0im      -0.000840021+0.0im
 -1.19678e-17+0.0im   2.28368e-17+0.0im      -0.000840021+0.0im
             ⋮                          ⋱
 -1.63665e-16+0.0im  -4.90423e-17+0.0im  …    -0.0128916+0.0im
 -9.39966e-18+0.0im   1.07219e-18+0.0im       -0.0109265+0.0im
 -1.42635e-16+0.0im  -5.75895e-17+0.0im      -0.00315567+0.0im
  3.09369e-17+0.0im  -1.12114e-16+0.0im       -0.0124533+0.0im
 -1.17169e-16+0.0im  -3.07707e-17+0.0im      -0.00222867+0.0im
 -1.74888e-16+0.0im  -5.44945e-17+0.0im  …    -0.0131631+0.0im
 -1.84581e-16+0.0im   1.04387e-16+0.0im      -0.00749321+0.0im
 -5.30365e-16+0.0im  -2.05309e-16+0.0im        -0.017154+0.0im
  4.65163e-17+0.0im    3.1328e-17+0.0im      -0.00717512+0.0im
```

```
 3.73715e-16+0.0im   1.64727e-16+0.0im      -0.00413608+0.0im
 -5.2381e-17+0.0im   5.51218e-18+0.0im  …   -0.00791887+0.0im
-2.47952e-18+0.0im  -2.93812e-18+0.0im      -0.00709532+0.0im
```

In [92]:

```
vec1 = vecP[:, size(D)[1]]
```

Out[92]:

```
2277-element Array{Complex{Float64},1}:
 -0.0008400208063672405 + 0.0im
  -0.014323097234788121 + 0.0im
 -0.0008400208063672552 + 0.0im
 -0.0008400208063672574 + 0.0im
 -0.0008400208063672631 + 0.0im
 -0.0024382486284503426 + 0.0im
 -0.0008400208063672422 + 0.0im
 -0.0008400208063672348 + 0.0im
 -0.0057124845438913194 + 0.0im
 -0.0008400208063672498 + 0.0im
 -0.0008400208063672521 + 0.0im
 -0.0008400208063672472 + 0.0im
 -0.0008400208063672558 + 0.0im
                         ⋮
  -0.012891638546080525 + 0.0im
  -0.010926538877798974 + 0.0im
    -0.0031556688356764 + 0.0im
  -0.012453258704987688 + 0.0im
 -0.0022286673298764132 + 0.0im
  -0.013163078039824683 + 0.0im
  -0.007493206677764491 + 0.0im
   -0.01715398181913199 + 0.0im
  -0.007175116622065958 + 0.0im
  -0.004136077879115737 + 0.0im
   -0.00791886602101884 + 0.0im
  -0.007095320018214208 + 0.0im
```

In [93]:

```
P^100000
```

Out[93]:

```
2277×2277 Array{Float64,2}:
 6.59658e-5   6.59658e-5   6.59658e-5   …  6.59658e-5   6.59658e-5
 0.00112477   0.00112477   0.00112477      0.00112477   0.00112477
 6.59658e-5   6.59658e-5   6.59658e-5      6.59658e-5   6.59658e-5
 6.59658e-5   6.59658e-5   6.59658e-5      6.59658e-5   6.59658e-5
 6.59658e-5   6.59658e-5   6.59658e-5      6.59658e-5   6.59658e-5
 0.000191473  0.000191473  0.000191473  …  0.000191473  0.000191473
 6.59658e-5   6.59658e-5   6.59658e-5      6.59658e-5   6.59658e-5
 6.59658e-5   6.59658e-5   6.59658e-5      6.59658e-5   6.59658e-5
 0.000448594  0.000448594  0.000448594     0.000448594  0.000448594
 6.59658e-5   6.59658e-5   6.59658e-5      6.59658e-5   6.59658e-5
 6.59658e-5   6.59658e-5   6.59658e-5   …  6.59658e-5   6.59658e-5
 6.59658e-5   6.59658e-5   6.59658e-5      6.59658e-5   6.59658e-5
 6.59658e-5   6.59658e-5   6.59658e-5      6.59658e-5   6.59658e-5
 ⋮                                     ⋱  ⋮
 0.00101236   0.00101236   0.00101236   …  0.00101236   0.00101236
 0.000858047  0.000858047  0.000858047     0.000858047  0.000858047
 0.000247811  0.000247811  0.000247811     0.000247811  0.000247811
 0.000977939  0.000977939  0.000977939     0.000977939  0.000977939
 0.000175014  0.000175014  0.000175014     0.000175014  0.000175014
 0.00103368   0.00103368   0.00103368   …  0.00103368   0.00103368
 0.000588432  0.000588432  0.000588432     0.000588432  0.000588432
 0.00134708   0.00134708   0.00134708      0.00134708   0.00134708
 0.000563453  0.000563453  0.000563453     0.000563453  0.000563453
 0.000324801  0.000324801  0.000324801     0.000324801  0.000324801
 0.000621858  0.000621858  0.000621858  …  0.000621858  0.000621858
 0.000557186  0.000557186  0.000557186     0.000557186  0.000557186
```

Scale the eigenvector corresponding to eigenvalue 1

We scale the eigenvector corresponding to eigenvalue 1 so that the sum of the components = 1

In [94]:

```
sumVec = 0
for i = 1:numPages
    sumVec = sumVec + vec1[i];
end
scaleFactor = 1 / sumVec
```

Out[94]:

```
-0.07842204792764866 - 0.0im
```

In [95]:

```
steadyStateVec = scaleFactor * vec1
```

Out[95]:

```
2277-element Array{Complex{Float64},1}:
   6.587615193715381e-5 + 0.0im
   0.001123246617818926 + 0.0im
   6.587615193715496e-5 + 0.0im
   6.587615193715514e-5 + 0.0im
   6.587615193715558e-5 + 0.0im
 0.00019121245079985639 + 0.0im
   6.587615193715394e-5 + 0.0im
   6.587615193715336e-5 + 0.0im
  0.0004479847366869973 + 0.0im
   6.587615193715454e-5 + 0.0im
   6.587615193715472e-5 + 0.0im
   6.587615193715434e-5 + 0.0im
     6.5876151937155e-5 + 0.0im
                        ⋮
  0.0010109886959266498 + 0.0im
  0.0008568815555580675 + 0.0im
  0.0002474740126752019 + 0.0im
  0.0009766100510179524 + 0.0im
 0.00017477665615835285 + 0.0im
   0.001032275536914511 + 0.0im
  0.0005876326132154239 + 0.0im
  0.0013452503843699828 + 0.0im
  0.0005626873396221251 + 0.0im
  0.0003243596976685017 + 0.0im
  0.0006210136906329679 + 0.0im
  0.0005564295265303996 + 0.0im
```

Create list of rankings

Create an n by 2 matrix R (n is the number of pages) where R_i1 is the page index and R_i2 is its corresponding probability.

In [96]:

```
R = zeros(numPages, 2)
```

Out[96]:

```
2277×2 Array{Float64,2}:
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 ⋮
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
 0.0  0.0
```

In [97]:

```
for i = 1:numPages
    setindex!(R, i, i, 1);
    setindex!(R, convert(Float64, steadyStateVec[i]), i, 2);
end
R
```

Out[97]:

```
2277×2 Array{Float64,2}:
    1.0  6.58762e-5
    2.0  0.00112325
    3.0  6.58762e-5
    4.0  6.58762e-5
    5.0  6.58762e-5
    6.0  0.000191212
    7.0  6.58762e-5
    8.0  6.58762e-5
    9.0  0.000447985
   10.0  6.58762e-5
   11.0  6.58762e-5
   12.0  6.58762e-5
   13.0  6.58762e-5
    ⋮
 2266.0  0.00101099
 2267.0  0.000856882
 2268.0  0.000247474
 2269.0  0.00097661
 2270.0  0.000174777
 2271.0  0.00103228
 2272.0  0.000587633
 2273.0  0.00134525
 2274.0  0.000562687
 2275.0  0.00032436
 2276.0  0.000621014
 2277.0  0.00055643
```

In [98]:

```
R = R[sortperm(R[:, 2]), :] # sort by the second column
```

Out[98]:

```
2277×2 Array{Float64,2}:
    8.0  6.58762e-5
  286.0  6.58762e-5
  571.0  6.58762e-5
 1141.0  6.58762e-5
    1.0  6.58762e-5
    7.0  6.58762e-5
   14.0  6.58762e-5
   12.0  6.58762e-5
   15.0  6.58762e-5
   19.0  6.58762e-5
   20.0  6.58762e-5
   21.0  6.58762e-5
   22.0  6.58762e-5
    ⋮
 2231.0  0.00760737
  925.0  0.00775298
 2111.0  0.00823065
 1357.0  0.00831825
 1975.0  0.00935199
 2250.0  0.0130232
  653.0  0.0141415
 2247.0  0.0182772
 2264.0  0.0214196
 1742.0  0.0277206
 1977.0  0.0304067
 1940.0  0.041486
```

In [99]:

```
ranks = Array{Int32}(undef, numPages);
for i = 1:numPages
    ranks[numPages + 1 - i] = R[i][1];
end
ranks
```

Out[99]:

```
2277-element Array{Int32,1}:
 1940
 1977
 1742
 2264
 2247
  653
 2250
 1975
 1357
 2111
  925
 2231
 1933
    ⋮
   21
   20
   19
   15
   12
   14
    7
    1
 1141
  571
  286
    8
```