Link Analysis: PageRank

CS246: Mining Massive Datasets

Jure Leskovec, Stanford University

http://cs246.stanford.edu

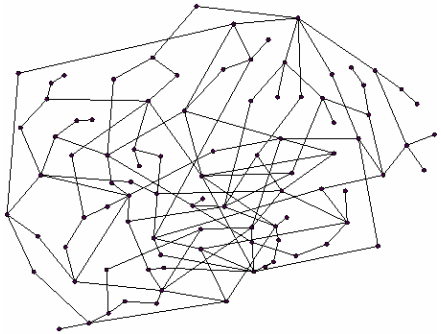
Ranking Nodes on the Graph

- **Web pages are not equally "important"**

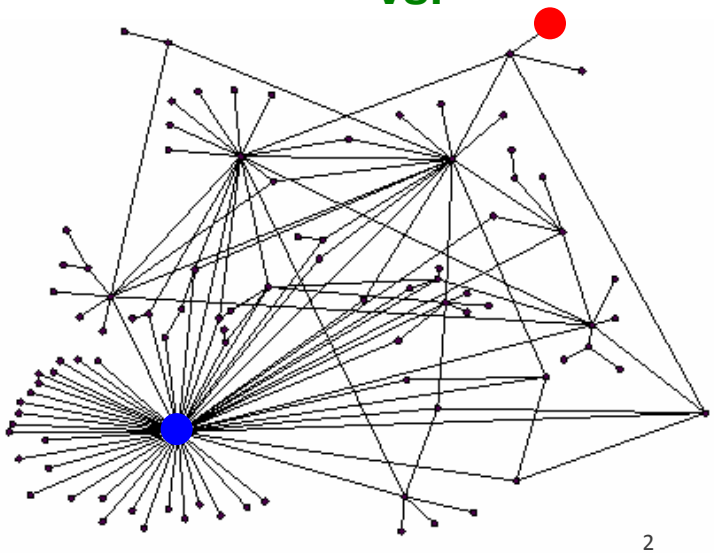
  www.joe-schmoe.com vs. www.stanford.edu

- **We already know:**

  Since there is large diversity in the connectivity of the webgraph we can **rank the pages by the link structure**



vs.



Link Analysis Algorithms

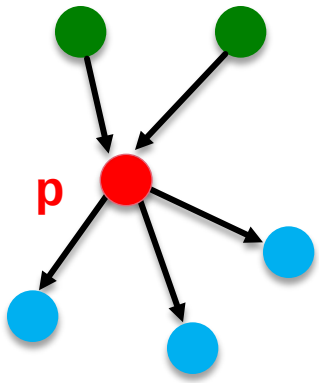
- We will cover the following Link Analysis approaches to computing importances of nodes in a graph:
  - Page Rank
  - Hubs and Authorities (HITS)
  - Topic-Specific (Personalized) Page Rank
  - Web Spam Detection Algorithms

Links as Votes

- Idea: Links as votes
  - **Page is more important if it has more links**
    - In-coming links? Out-going links?
- **Think of in-links as votes:**
  - www.stanford.edu has 23,400 inlinks
  - www.joe-schmoe.com has 1 inlink
- Are all in-links are equal?
  - Links from important pages count more
  - Recursive question!

Simple Recursive Formulation

- Each link's vote is proportional to the **importance** of its source page
- If page *p* with importance *x* has *n* out-links, each link gets *x/n* votes
- Page *p*'s own importance is the sum of the votes on its in-links

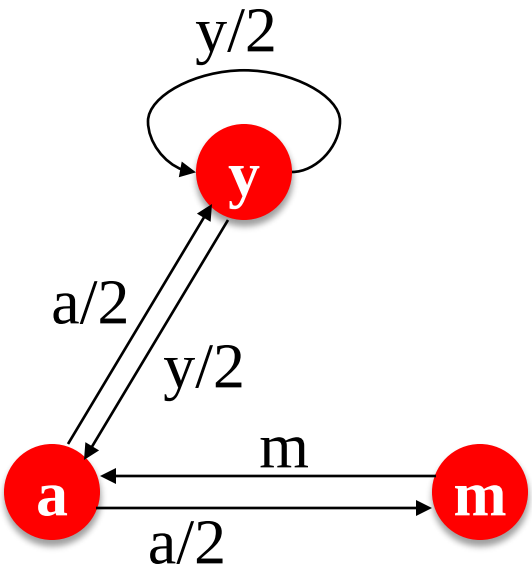


PageRank: The "Flow" Model

- A "vote" from an important page is worth more
- A page is important if it is pointed to by other important pages
- **Define a "rank" r_j for node j**

$$r_j = \sum_{i \to j} \frac{r_i}{d_{out}(i)}$$

The web in 1839



Flow equations:

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

Solving the Flow Equations

Flow equations:

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

- **3 equations, 3 unknowns, no constants**
  - No unique solution
  - All solutions equivalent modulo scale factor
- **Additional constraint forces uniqueness**
  - $y + a + m = 1$
  - $y = 2/5,\ a = 2/5,\ m = 1/5$
- Gaussian elimination method works for small examples, but we need a better method for large web-size graphs

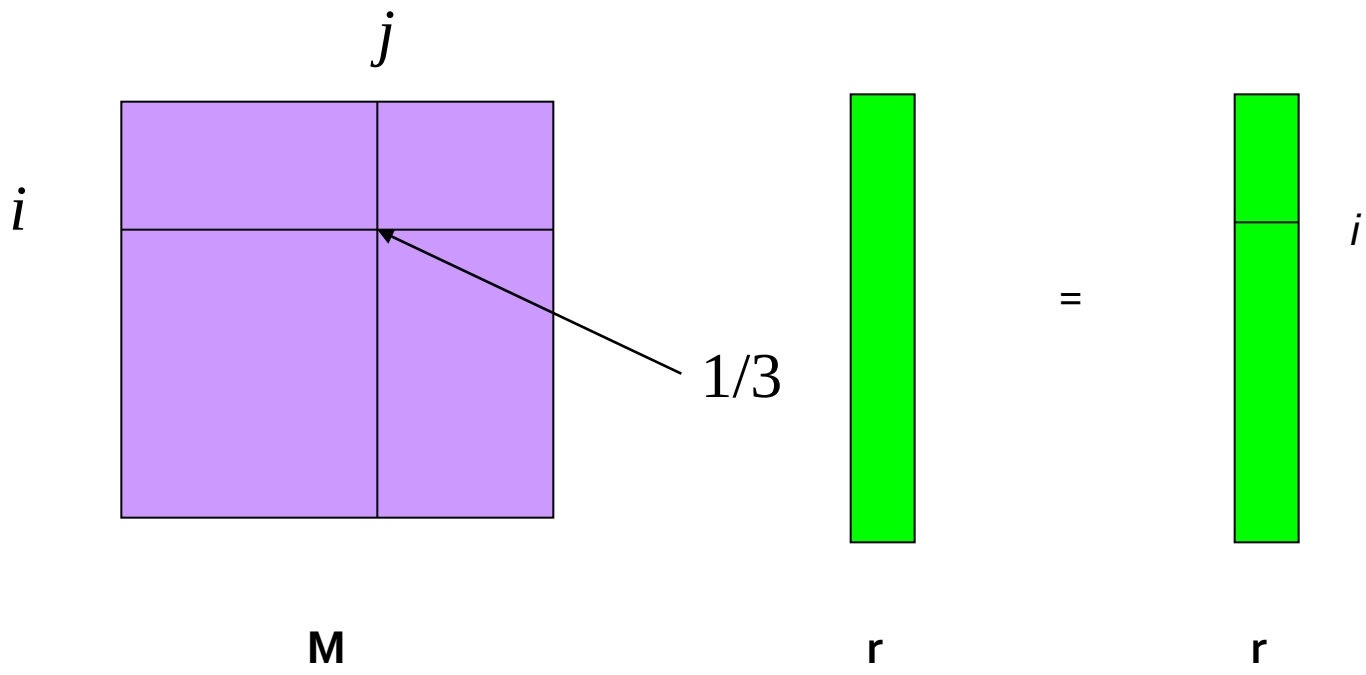
PageRank: Matrix Formulation

- **Stochastic adjacency matrix M**
  - Let page j has d_j out-links
  - If $j \rightarrow i$, then $M_{ij} = 1/d_j$ else $M_{ij} = 0$
    - M is a **column stochastic matrix**
      - Columns sum to 1
- **Rank vector r:** vector with an entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$
- **The flow equations can be written**

$$r = M\,r$$

Example

- Suppose page *j* links to 3 pages, including *i*



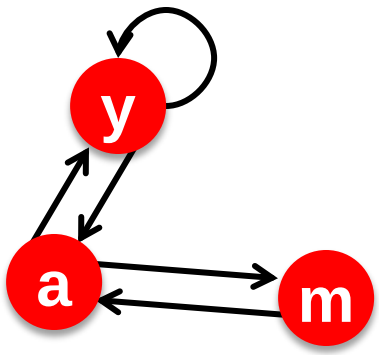
Eigenvector Formulation

- **The flow equations can be written**

$$r = M \cdot r$$

- So the rank vector is an eigenvector of the stochastic web matrix
  - In fact, its first or principal eigenvector, with corresponding eigenvalue 1

Example



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

r = Mr

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

Power Iteration Method

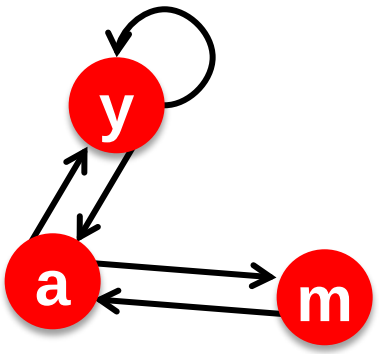
- **Given a web graph with n nodes, where the nodes are pages and edges are hyperlinks**
- **Power iteration:** a simple iterative scheme
  - Suppose there are *N* web pages
  - Initialize: $\mathbf{r}^{(0)} = [1/N,\ldots,1/N]^T$
  - Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
  - Stop when $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$
    - $|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the L_1 norm
    - Can use any other vector norm e.g., Euclidean

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

d_i …. out-degree of node i

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate
    - $r_i = \sum_j M_{ij} \cdot r_j$
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

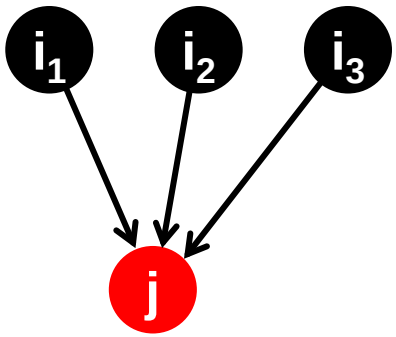
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 6/15 \\ 1/3 & 3/6 & 1/3 & 11/24 & \ldots & 6/15 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 3/15 \end{matrix}$$

Iteration 0, 1, 2, …

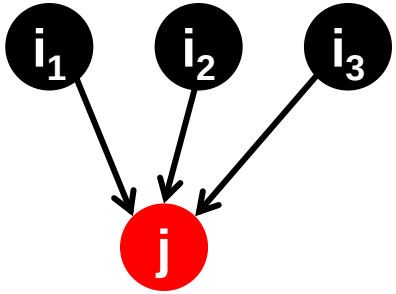
Random Walk Interpretation

- **Imagine a random web surfer:**
  - At any time t, surfer is on some page u
  - At time $t+1$, the surfer follows an out-link from u uniformly at random
  - Ends up on some page v linked from u
  - Process repeats indefinitely
- **Let:**
  - $\boldsymbol{p}(t)$ … vector whose i^{th} coordinate is the prob. that the surfer is at page i at time t
  - $\boldsymbol{p}(t)$ is a probability distribution over pages



$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_{\mathrm{out}}(\mathrm{i})}$$

The Stationary Distribution



$$p(t+1) = \mathbf{M} \cdot p(t)$$

- **Where is the surfer at time *t+1*?**
  - Follows a link uniformly at random

$$\boldsymbol{p}(t+1) = \boldsymbol{M} \cdot \boldsymbol{p}(t)$$

- Suppose the random walk reaches a state

$$\boldsymbol{p}(t+1) = \boldsymbol{M} \cdot \boldsymbol{p}(t) = \boldsymbol{p}(t)$$

then $\boldsymbol{p}(t)$ is stationary distribution of a random walk

- **Our rank vector** $\boldsymbol{r}$ satisfies $\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$
  - So, it is a stationary distribution for the random walk

PageRank: Three Questions

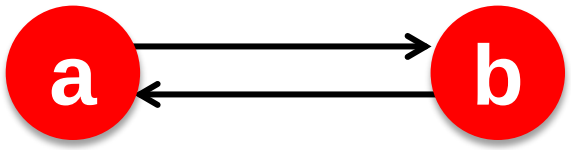
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

or equivalently

$$r = Mr$$

- **Does this converge?**
- **Does it converge to what we want?**
- **Are results reasonable?**

Does This Converge?



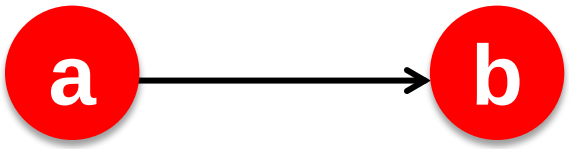
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

- **Example:**

$$\begin{matrix} r_a \\ r_b \end{matrix} = \begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}$$

Iteration 0, 1, 2, …

Does it Converge to What We Want?



$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

- **Example:**

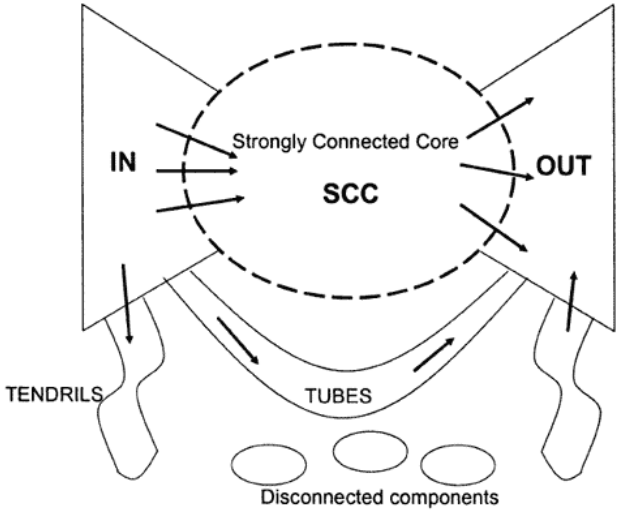
$$\begin{matrix} r_a \\ r_b \end{matrix} = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

Iteration 0, 1, 2, …

Problems with the "Flow" Model

2 problems:

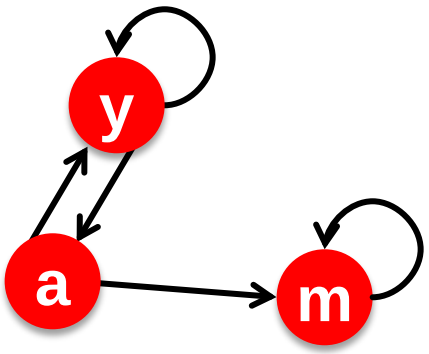
- Some pages are "**dead ends**" (have no out-links)
  - Such pages cause importance to "leak out"



- **Spider traps** (all out links are within the group)
  - Eventually spider traps absorb all importance

Problem: Spider Traps

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2$$
$$r_m = r_a/2 + r_m$$

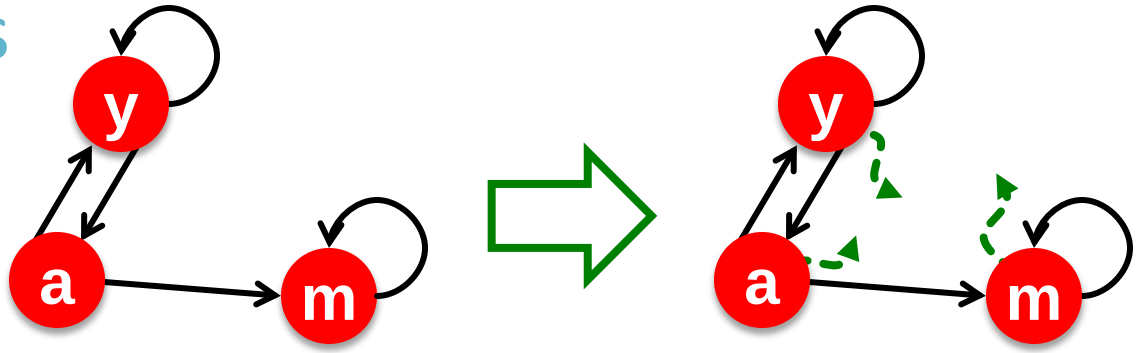
- **Example:**

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

Iteration 0, 1, 2, …

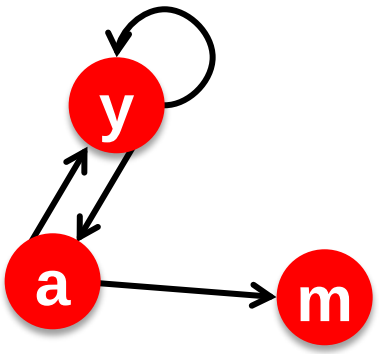
Solution: Random Teleports

- **The Google solution for spider traps:** At each time step, the random surfer has two options:
  - With probability β, follow a link at random
  - With probability 1-β, jump to some page uniformly at random
  - Common values for β are in the range 0.8 to 0.9
- Surfer will teleport out of spider trap within a few time steps



Problem: Dead Ends

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

$r_y = r_y/2 + r_a/2$
$r_a = r_y/2$
$r_m = r_a/2$

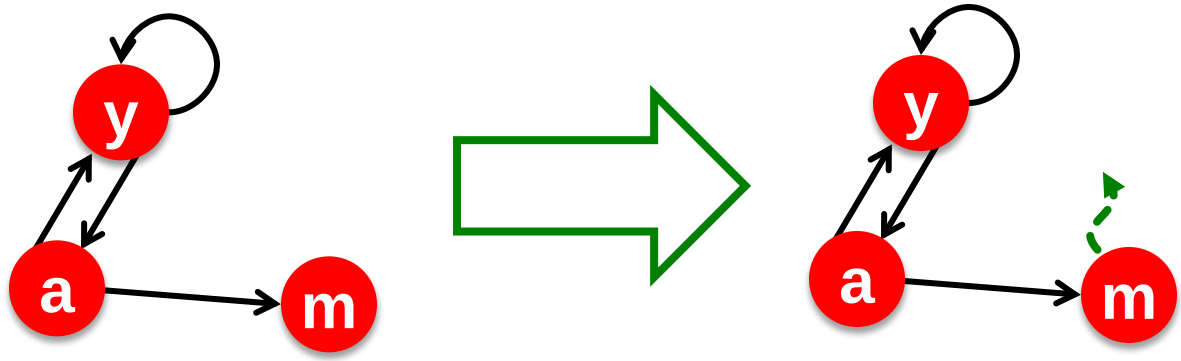
- **Example:**

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

Iteration 0, 1, 2, …

Solution: Dead Ends

- **Teleports:** Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

Why Teleports Solve the Problem?

$$r^{(t+1)} = Mr^{(t)}$$

Markov Chains

- Set of states X
- Transition matrix P where $P_{ij} = P(X_t=i \mid X_{t-1}=j)$
- π specifying the probability of being at each state $x \in X$
- Goal is to find π such that π = P π

Why is This Analogy Useful?

- **Theory of Markov chains**

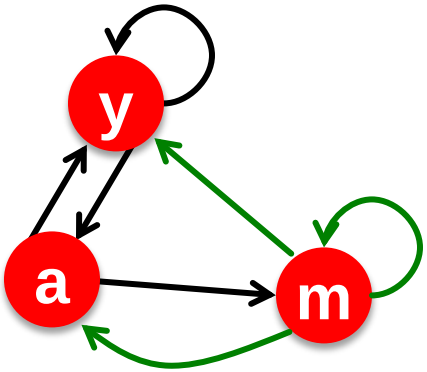
- Fact: For any start vector, the power method applied to a Markov transition matrix P will converge to a unique positive stationary vector as long as P is **stochastic**, **irreducible** and **aperiodic**.

Make M Stochastic

- **Stochastic**: Every column sums to 1
- **A possible solution:** Add green links

$$S = M + a^T \left(\frac{1}{n}\mathbf{1}\right)$$

- a_i…=1 if node *i* has out deg 0, =0 else
- **1**…vector of all 1s

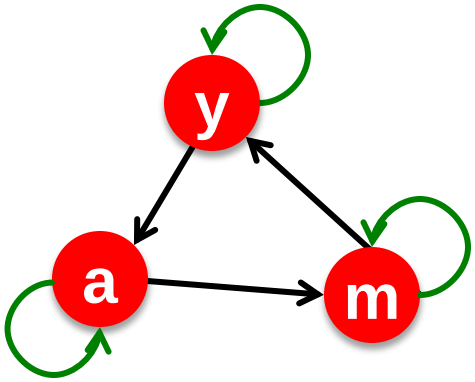


	y	a	m
y	½	½	**1/3**
a	½	0	**1/3**
m	0	½	**1/3**

$$\mathbf{r_y = r_y/2 + r_a/2 + r_m/3}$$
$$\mathbf{r_a = r_y/2 + r_m/3}$$
$$\mathbf{r_m = r_a/2 + r_m/3}$$

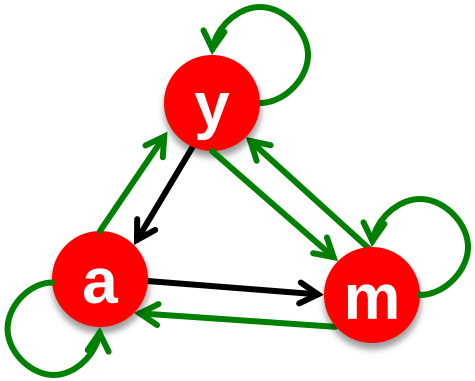
Make M Aperiodic

- A chain is **periodic** if there exists $k > 1$ such that the interval between two visits to some state s is always a multiple of k.
- **A possible solution:** Add green links



Make M Irreducible

- From any state, there is a non-zero probability of going from any one state to any another
- **A possible solution:** Add green links



Solution: Random Jumps

- **Google's solution that does it all:**
  - Makes ***M*** stochastic, aperiodic, irreducible
- At each step, random surfer has two options:
  - With probability $1-\beta$, follow a link at random
  - With probability β, jump to some random page
- **PageRank equation** [Brin-Page, 98]

$$r_j = \sum_{i \to j} (1-\beta)\frac{r_i}{d_i} + \beta\frac{1}{n}$$

Assuming we follow random teleport links with probability 1.0 from dead-ends

d_i ... out-degree of node i

The Google Matrix

- **PageRank equation** [Brin-Page, 98]

$$r_j = \sum_{i \to j} (1-\beta)\frac{r_i}{d_i} + \beta\frac{1}{n}$$

- **The Google Matrix *A*:**

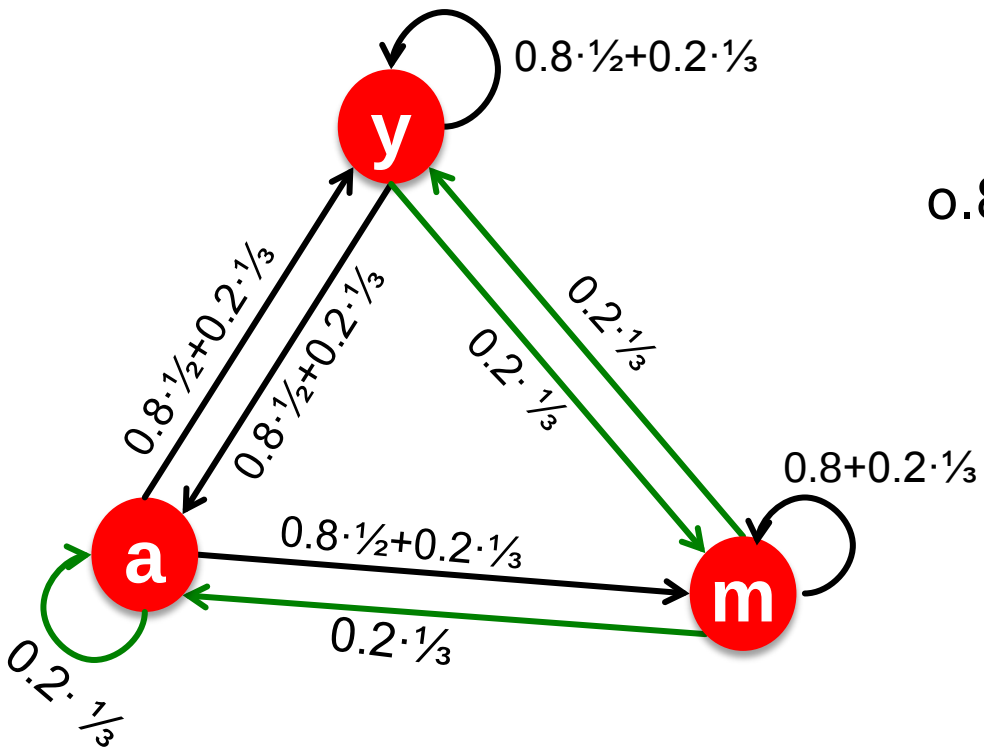
$$A = (1-\beta)S + \beta\frac{1}{n}\mathbf{1}\cdot\mathbf{1}^T$$

- ***G* is stochastic, aperiodic and irreducible, so**

$$r^{(t+1)} = A \cdot r^{(t)}$$

- **What is β ?**
  - In practice $\beta = 0.15$ (make 5 steps and jump)

Random Teleports ($\beta = 0.8$)



S **$1/n \cdot 1 \cdot 1^T$**

$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$\begin{array}{c} y \\ a \\ m \end{array} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

A

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{ccccc} 1/3 & 0.33 & 0.24 & 0.26 & \\ 1/3 & 0.20 & 0.20 & 0.18 & \ldots \\ 1/3 & 0.46 & 0.52 & 0.56 & \end{array} \begin{array}{c} \\ 7/33 \\ 5/33 \\ 21/33 \end{array}$$

Matrix Formulation

- Suppose there are N pages
- Consider a page j, with set of out-links $O(j)$
- We have $\boldsymbol{M}_{ij} = 1/|O(j)|$ when $j \rightarrow i$ and $\boldsymbol{M}_{ij} = 0$ otherwise
- **The random teleport is equivalent to**
  - Adding a teleport link from j to every other page with probability $(1-\beta)/N$
  - Reducing the probability of following each out-link from $1/|O(j)|$ to $\beta/|O(j)|$
  - **Equivalent:** Tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

Page Rank

- Construct the N x N matrix ***A*** as follows
  - $\boldsymbol{A}_{ij} = \beta \cdot \boldsymbol{M}_{ij} + (1-\beta)/N$
- Verify that **A** is a stochastic matrix
- The page rank vector ***r*** is the principal eigenvector of this matrix ***A***
  - satisfying $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}$
- Equivalently, ***r*** is the stationary distribution of the random walk with teleports

Computing Page Rank

- **Key step is matrix-vector multiplication**
  - $r^{new} = A \cdot r^{old}$
- Easy if we have enough main memory to hold **A**, $\mathbf{r}^{old}$, $\mathbf{r}^{new}$
- **Say N = 1 billion pages**
  - We need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - Matrix **A** has N^2 entries
    - 10^{18} is a large number!

$$\mathbf{A} = \beta \cdot \mathbf{M} + (1-\beta)\,[1/N]_{N \times N}$$

$$A = 0.8 \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 0 \\ 0 & ½ & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$= \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

Rearranging the Equation

- $r = A \cdot r$, where $A_{ij} = \beta\, M_{ij} + \frac{1-\beta}{N}$
- $r_i = \sum_{j=1}^{N} A_{ij} \cdot r_j$
- $r_i = \sum_{j=1}^{N} \left[\beta\, M_{ij} + \frac{1-\beta}{N}\right] \cdot r_j$

  $= \sum_{j=1}^{N} \beta\, M_{ij} \cdot r_j + \frac{1-\beta}{N} \sum_{j=1}^{N} r_j$

  $= \sum_{j=1}^{N} \beta\, M_{ij} \cdot r_j + \frac{1-\beta}{N}$, since $\sum r_j = 1$
- So, $r = \beta\, M \cdot r + \left[\frac{1-\beta}{N}\right]_N$

$[x]_N$ … a vector of length N with all entries x

Sparse Matrix Formulation

- We can rearrange the **PageRank equation**
  - $\boldsymbol{r} = \boldsymbol{\beta M} \cdot \boldsymbol{r} + \left[\frac{\mathbf{1}-\boldsymbol{\beta}}{\boldsymbol{N}}\right]_{\boldsymbol{N}}$
    - $[(1-\beta)/N]_N$ is an N-vector with all entries $(1-\beta)/N$
- ***M*** is a **sparse matrix!**
  - 10 links per node, approx 10N entries
- So in each iteration, we need to:
  - Compute $\boldsymbol{r}^{new} = \beta \boldsymbol{M} \cdot \boldsymbol{r}^{old}$
  - Add a constant value $(1-\beta)/N$ to each entry in $\boldsymbol{r}^{new}$

Sparse Matrix Encoding

- **Encode sparse matrix using only nonzero entries**
  - Space proportional roughly to number of links
  - Say 10N, or 4*10*1 billion = 40GB
  - **Still won't fit in memory, but will fit on disk**

source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

Basic Algorithm: Update Step

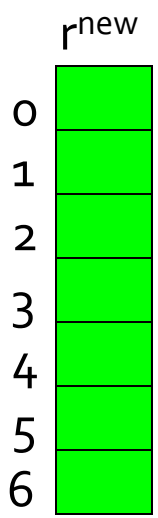
- Assume enough RAM to fit r^{new} into memory
  - Store r^{old} and matrix **M** on disk
- **Then 1 step of power-iteration is:**

Initialize all entries of r^{new} to $(1-\beta)/N$

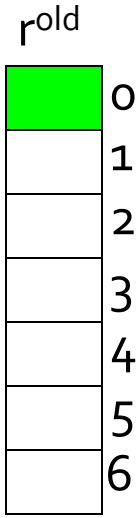
For each page *p* (of out-degree *n*):

Read into memory: p, n, $dest_1,\ldots,dest_n$, $r^{old}(p)$

for j = 1…n: $r^{new}(dest_j) \mathrel{+}= \beta\, r^{old}(p) / n$



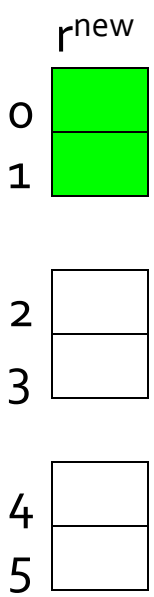
src	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23



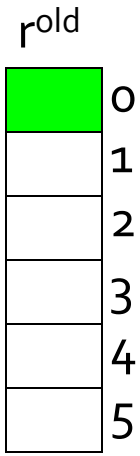
Analysis

- Assume enough RAM to fit $\boldsymbol{r}^{new}$ into memory
  - Store $\boldsymbol{r}^{old}$ and matrix **M** on disk
- In each iteration, we have to:
  - Read $\boldsymbol{r}^{old}$ and **M**
  - Write $\boldsymbol{r}^{new}$ back to disk
  - IO cost = $2|\boldsymbol{r}| + |\boldsymbol{M}|$
- **Question:**
  - What if we could not even fit $\mathbf{r}^{new}$ in memory?

Block-based Update Algorithm



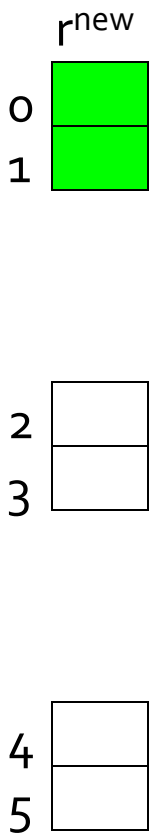
src	degree	destination
0	4	0, 1, 3, 5
1	2	0, 5
2	2	3, 4



Analysis of Block Update

- Similar to nested-loop join in databases
  - Break $\boldsymbol{r}^{new}$ into k blocks that fit in memory
  - Scan $\boldsymbol{M}$ and $\boldsymbol{r}^{old}$ once for each block
- k scans of $\boldsymbol{M}$ and $\boldsymbol{r}^{old}$
  - $k(|\boldsymbol{M}| + |\boldsymbol{r}|) + |\boldsymbol{r}| = k|\boldsymbol{M}| + (k+1)|\boldsymbol{r}|$
- **Can we do better?**
  - Hint: $\boldsymbol{M}$ is much bigger than $\boldsymbol{r}$ (approx 10-20x), so we must avoid reading it k times per iteration

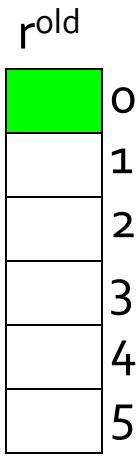
Block-Stripe Update Algorithm



src	degree	destination
0	4	0, 1
1	3	0
2	2	1

0	4	3
2	2	3

0	4	5
1	3	5
2	2	4



Block-Stripe Analysis

- Break $\boldsymbol{M}$ into stripes
  - Each stripe contains only destination nodes in the corresponding block of $\boldsymbol{r}^{new}$
- Some additional overhead per stripe
  - But it is usually worth it
- Cost per iteration
  - $|\boldsymbol{M}|(1+\varepsilon) + (k+1)|\boldsymbol{r}|$

Some Problems with Page Rank

- **Measures generic popularity of a page**
  - Biased against topic-specific authorities
  - **Solution:** Topic-Specific PageRank (next)
- **Uses a single measure of importance**
  - Other models e.g., hubs-and-authorities
  - **Solution:** Hubs-and-Authorities (next)
- **Susceptible to Link spam**
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank (next)