Analysis of Large Graphs: Link Analysis, PageRank

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Charilaos Kanatsoulis, Stanford University
http://cs246.stanford.edu

New Topic: Graph Data!

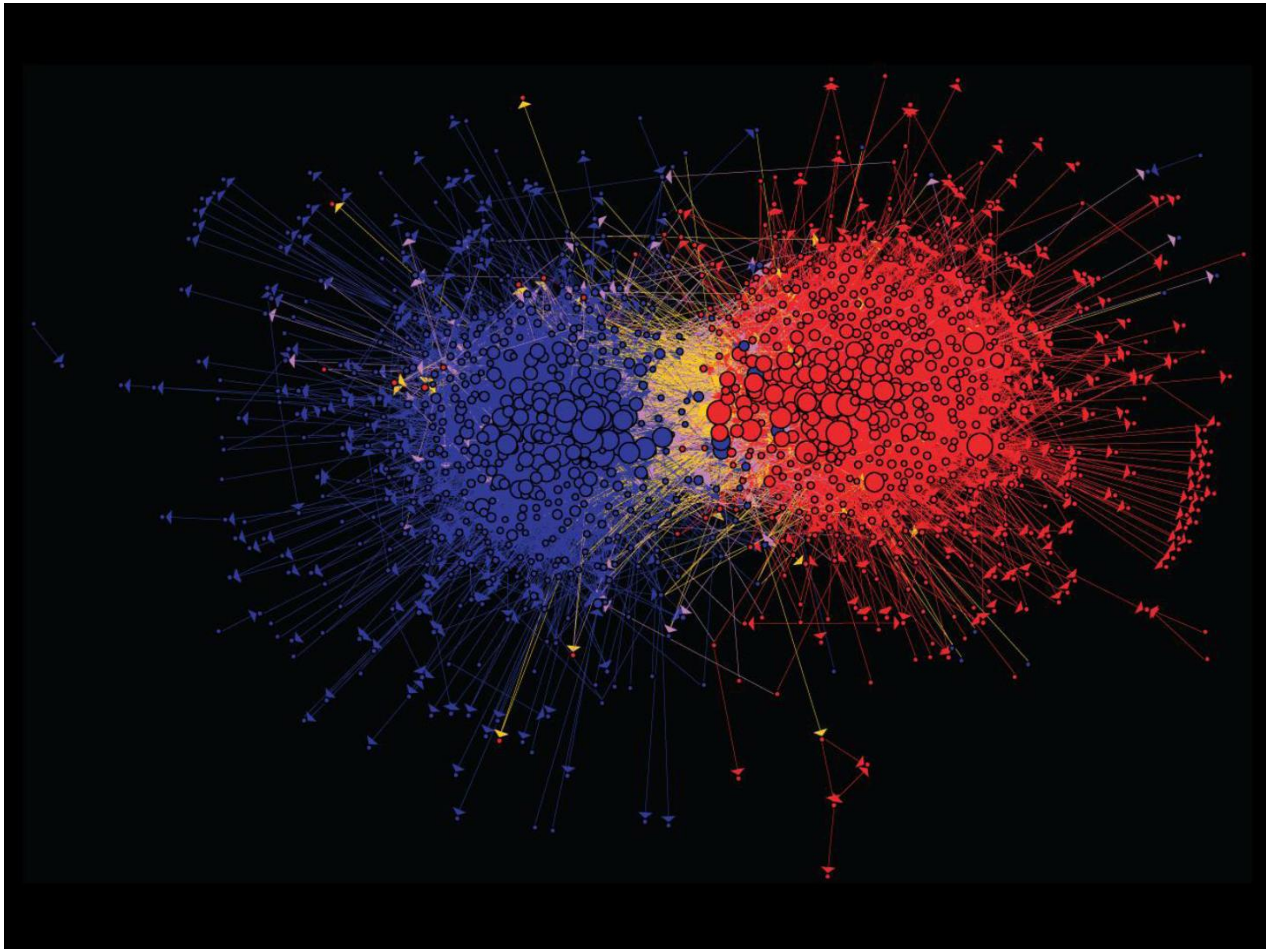
High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	**PageRank,** SimRank	Filtering data streams	SVM	Recommen der systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensional ity reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Graph Data: Social Networks

Facebook social graph

4-degrees of separation [Backstrom-Boldi-Rosa-Ugander-Vigna, 2011]

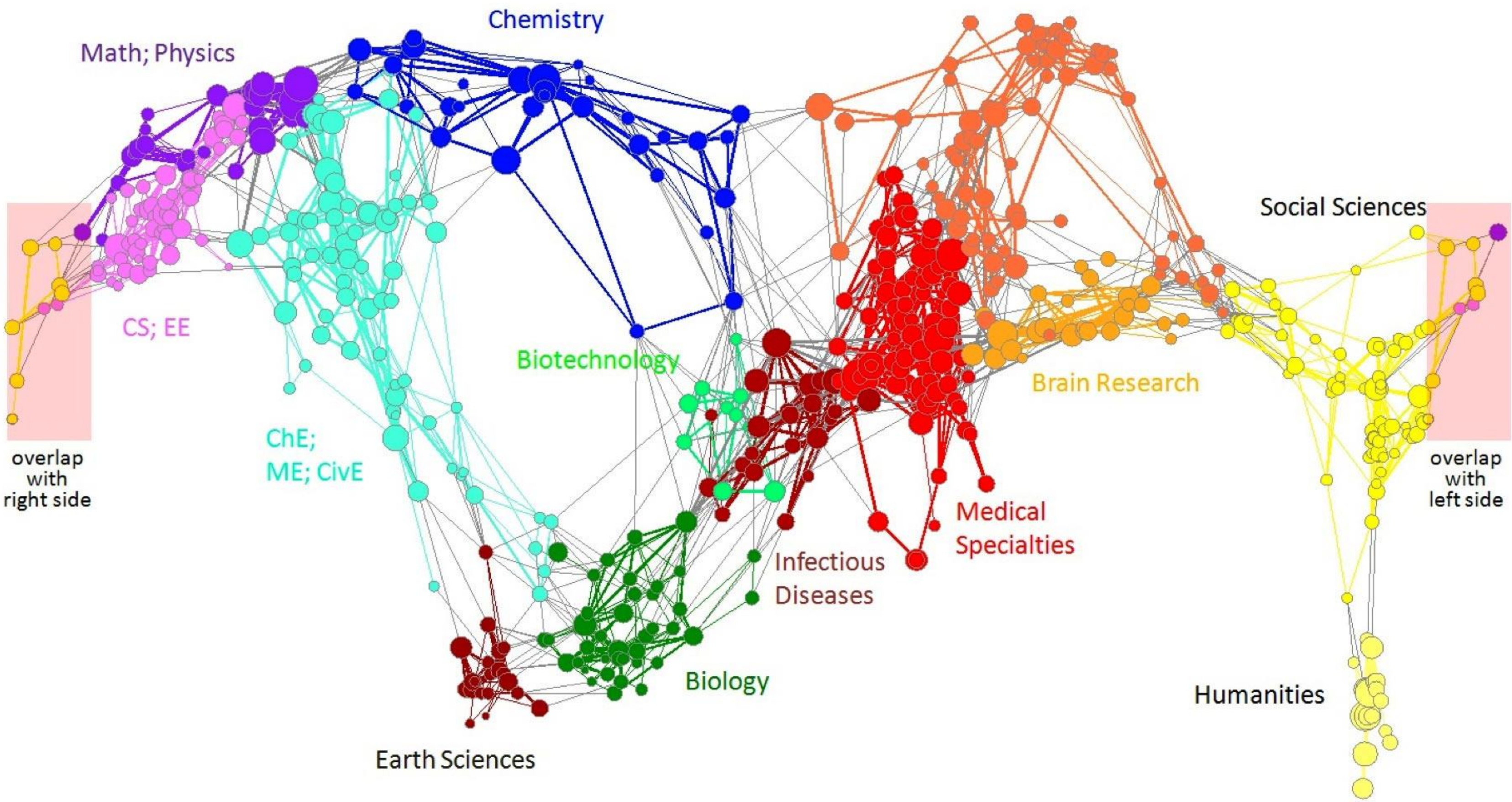
Graph Data: Media Networks



Connections between political blogs

Polarization of the network [Adamic-Glance, 2005]

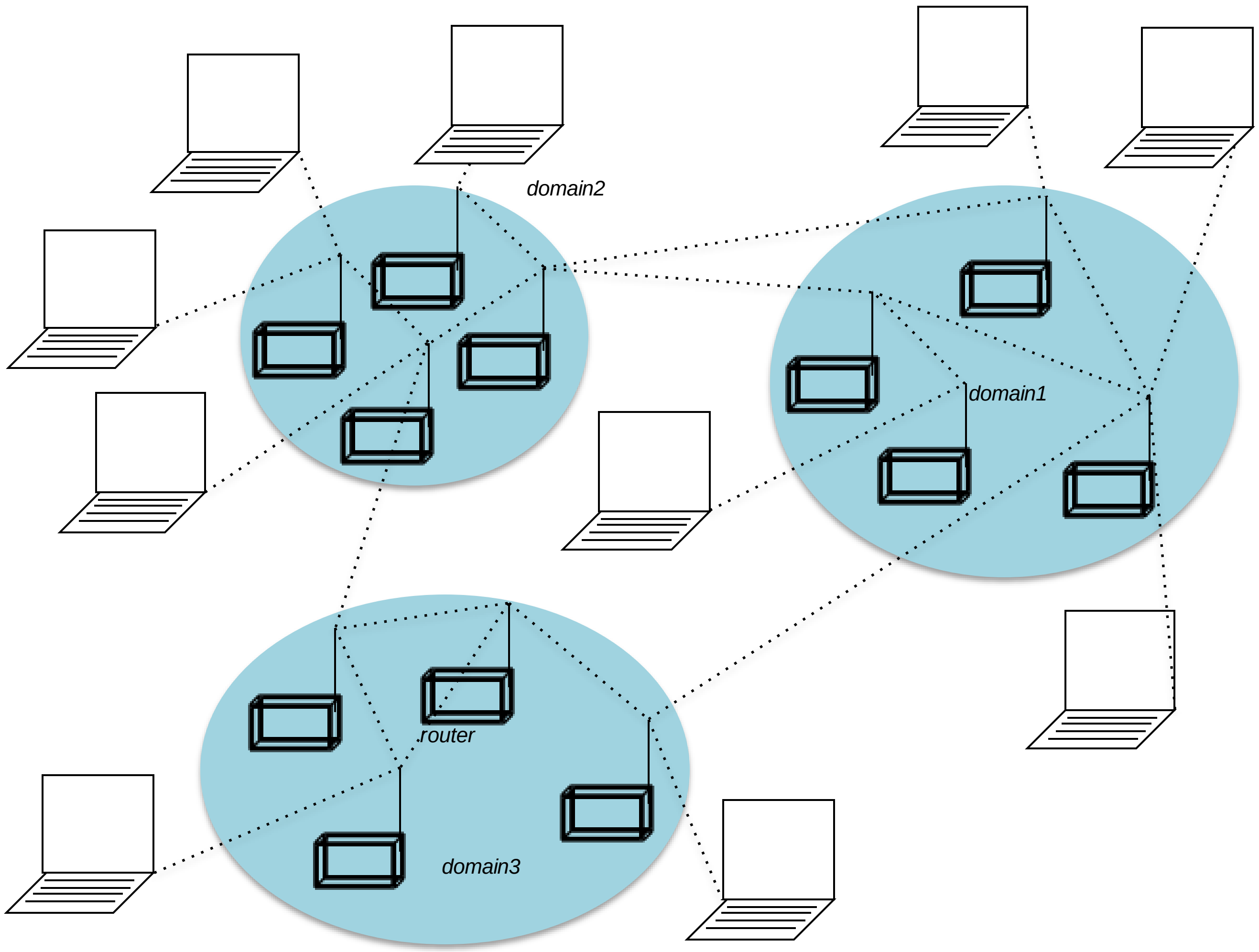
Graph Data: Information Nets



Citation networks and Maps of science

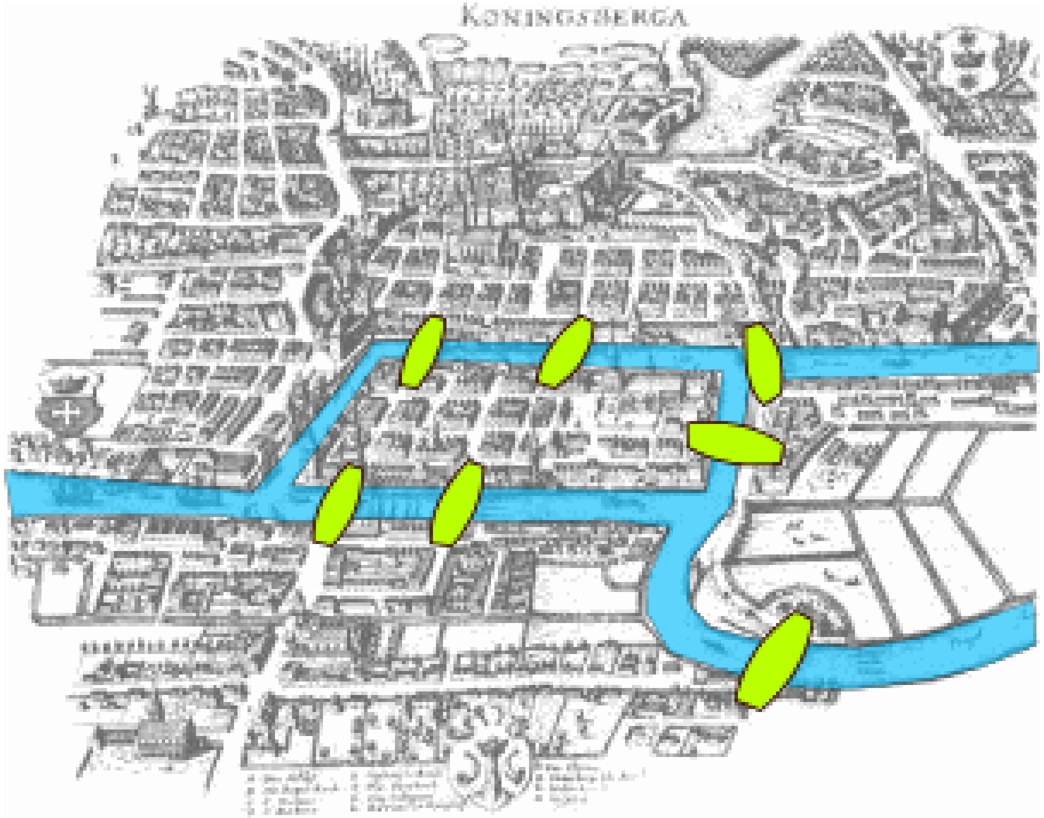
[Börner et al., 2012]

Graph Data: Communication Networks



Internet

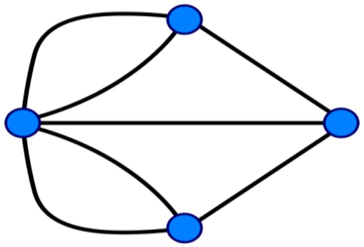
Graph Data: Technological Networks



Seven Bridges of Königsberg

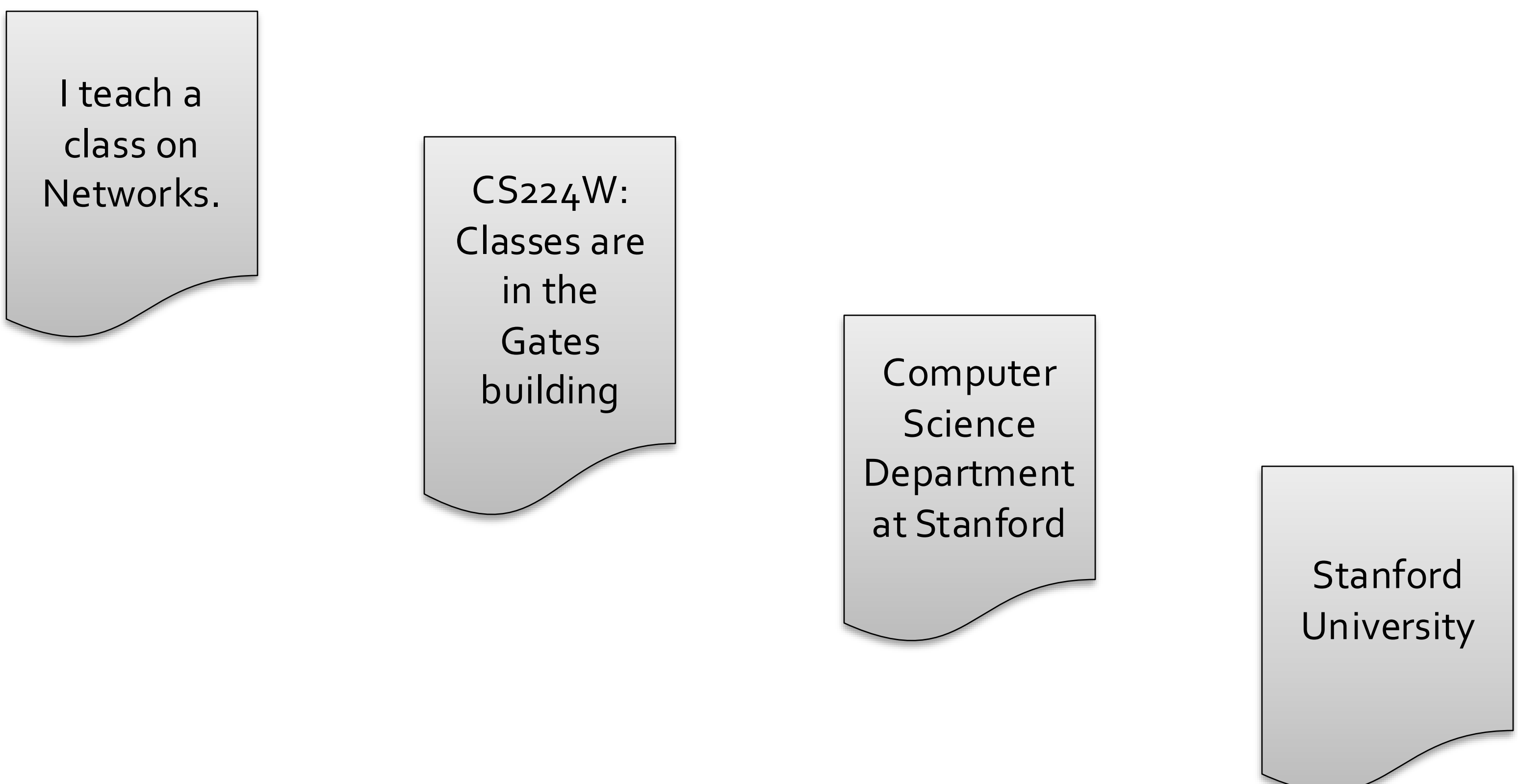
[Euler, 1735]

Return to the starting point by traveling each link of the graph once and only once.



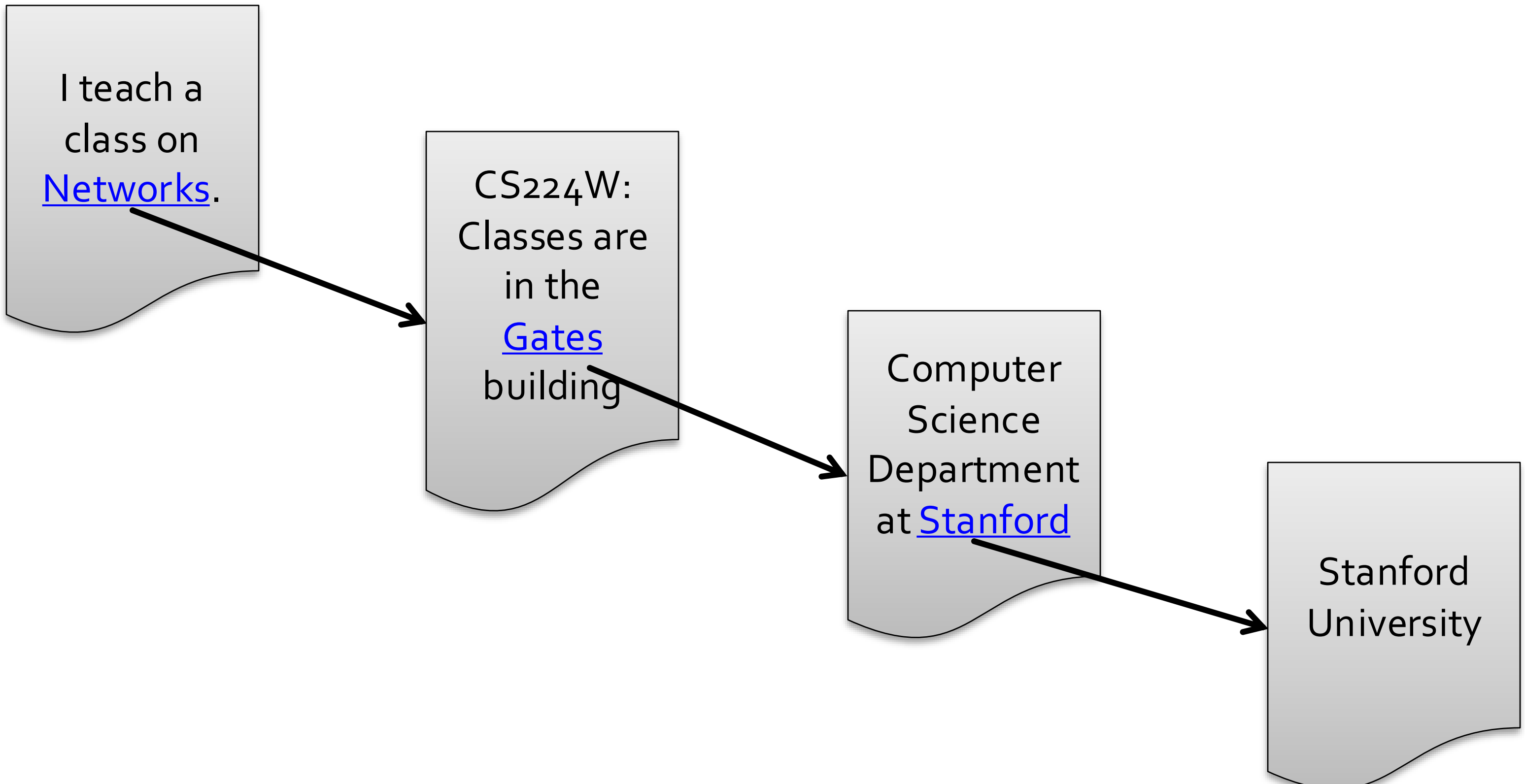
Web as a Graph

- **Web as a directed graph:**
  - **Nodes: Webpages**
  - **Edges: Hyperlinks**

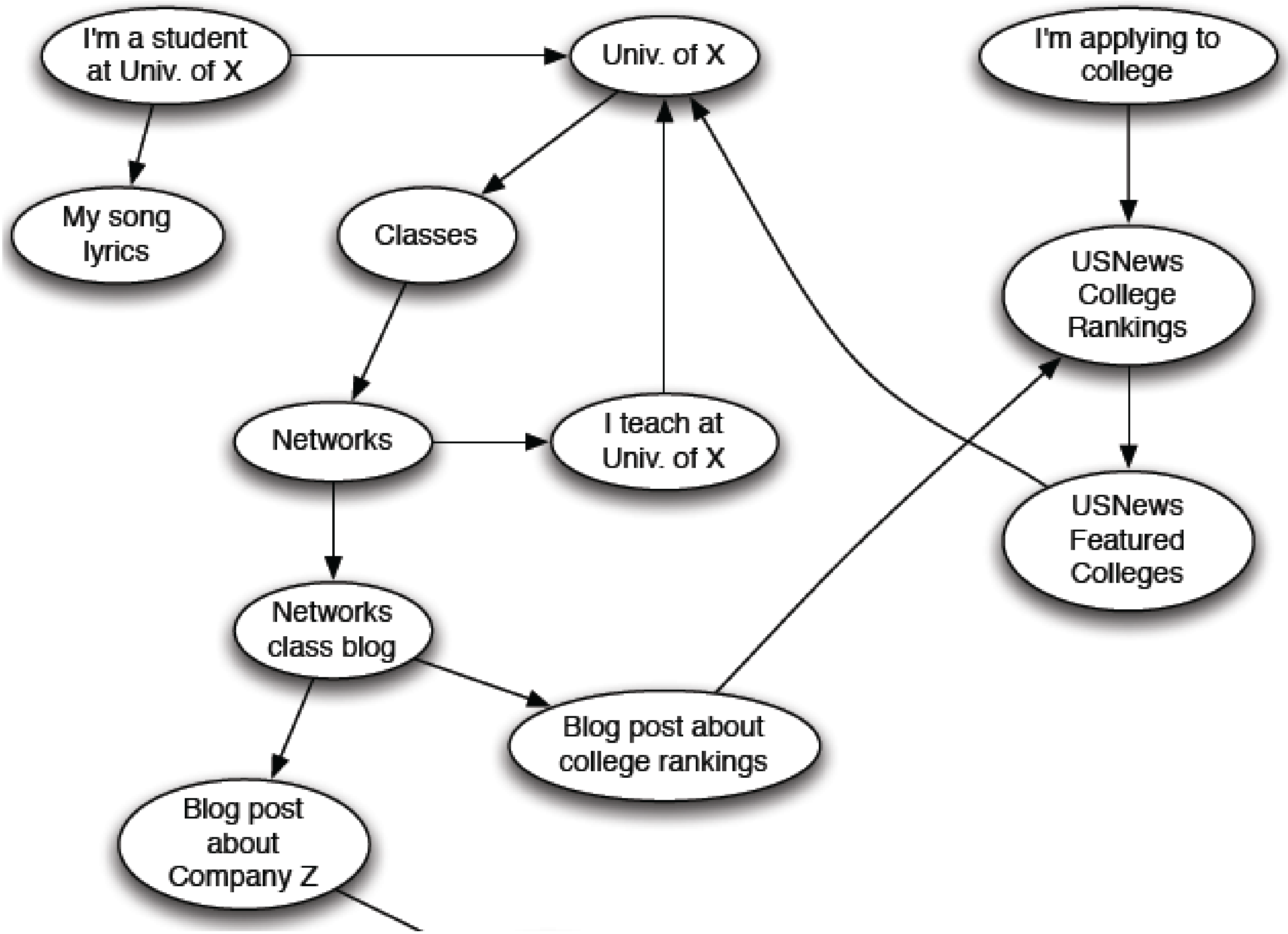


Web as a Graph

- **Web as a directed graph:**
  - **Nodes: Webpages**
  - **Edges: Hyperlinks**



Web as a Directed Graph



Broad Question

- **How to organize the Web?**
- **First try:** Human curated **Web directories**
  - Yahoo, DMOZ, LookSmart
- **Second try: Web Search**
  - **Information Retrieval** investigates: Find relevant docs in a small and trusted set
    - Newspaper articles, Patents, etc.
  - **But:** Web is **huge**, full of untrusted documents, random things, web spam, etc.

Web Search: 2 Challenges

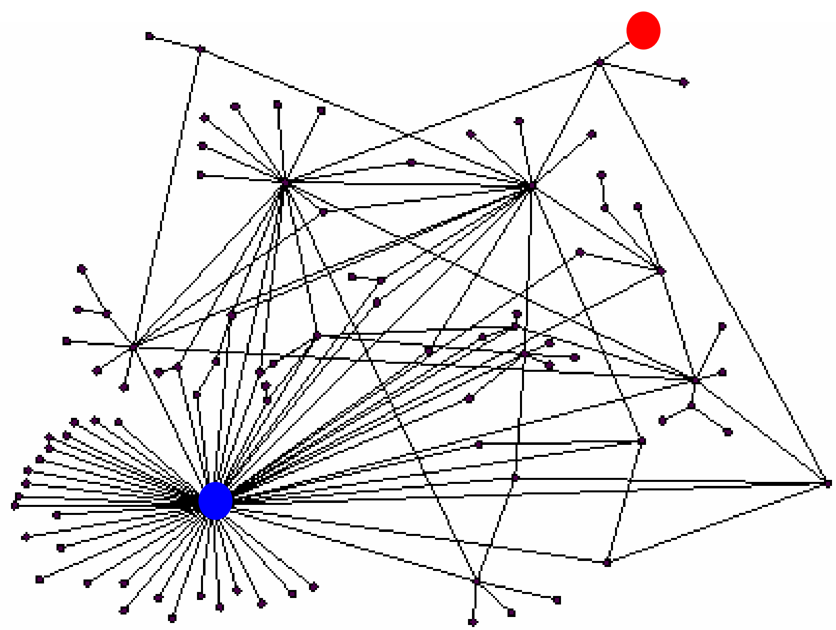
2 challenges of web search:

- **(1) Web contains many sources of information Who to "trust"?**
  - **Trick:** Trustworthy pages may point to each other!
- **(2) What is the "best" answer to query "newspaper"?**
  - No single right answer
  - **Trick:** Pages that actually know about newspapers might all be pointing to many newspapers

Ranking Nodes on the Graph

- **All web pages are not equally "important"**
  thispersondoesnotexist.com vs. www.stanford.edu

- There is a large diversity in the web-graph node connectivity.
  Let's rank the pages by the link structure!



Link Analysis Algorithms

- We will cover the following **Link Analysis approaches** for computing **importance of nodes in a graph**:
  - PageRank
  - Topic-Specific (Personalized) PageRank
  - Web Spam Detection Algorithms

PageRank:
The “Flow” Formulation

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 **millions** in-links
  - thispersondoesnotexist.com has a **few thousands** in-link
- **Are all in-links equal?**
  - **Links from important pages count more**
  - Recursive question!

Intuition – (1)

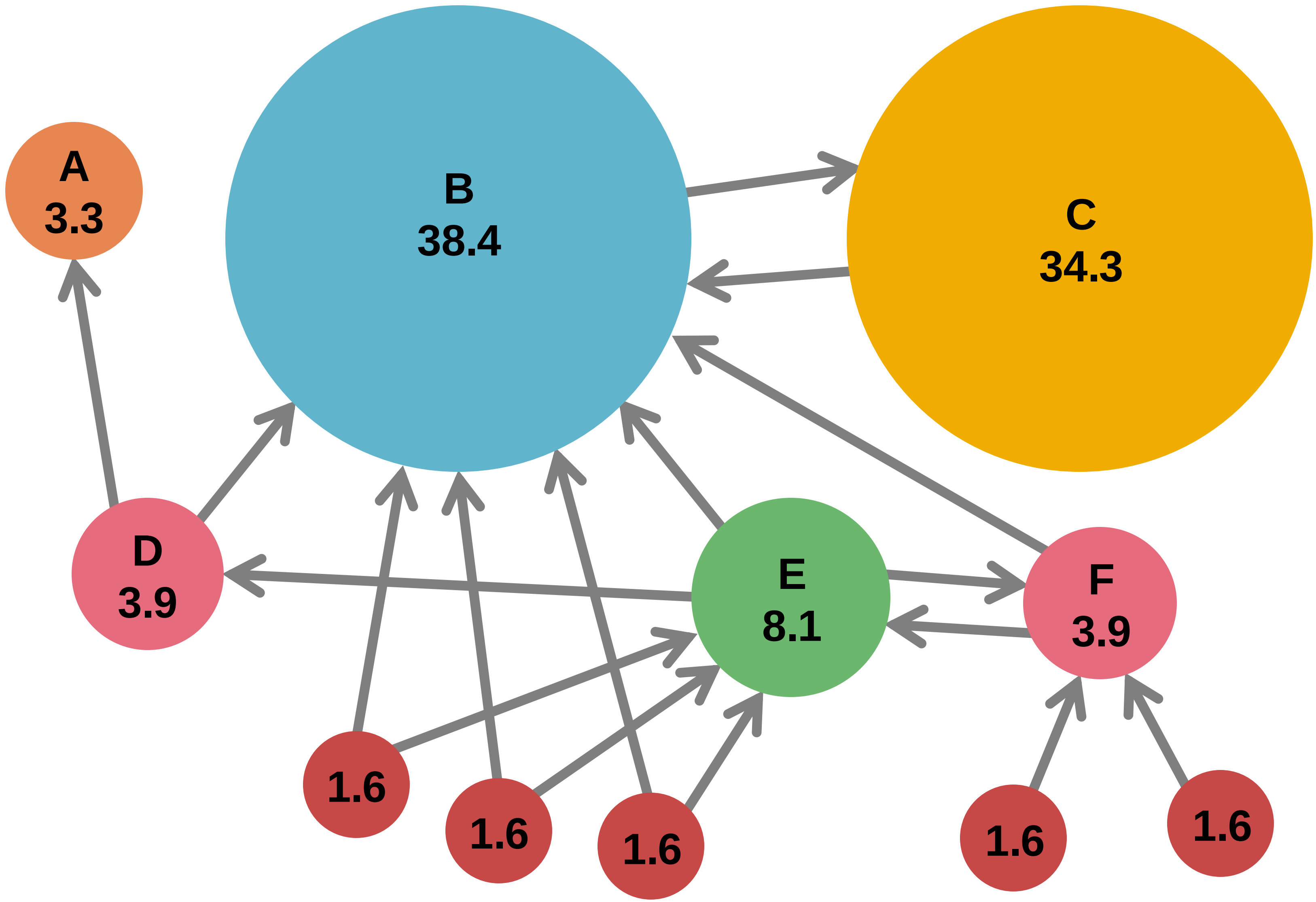
- Web pages are important if people visit them a lot.
- But we can't watch everybody using the Web.
- A good surrogate for visiting pages is to assume people follow links randomly.
- Leads to *random surfer* model:
  - Start at a random page and follow random out-links repeatedly, from whatever page you are at.
  - *PageRank* = limiting probability of being at a page.

Intuition – (2)

- **Solve the recursive equation:** "importance of a page = its share of the importance of each of its predecessor pages"
  - Equivalent to the random-surfer definition of PageRank

- Technically, *importance* = the principal eigenvector of the transition matrix of the Web
  - A few fix-ups needed

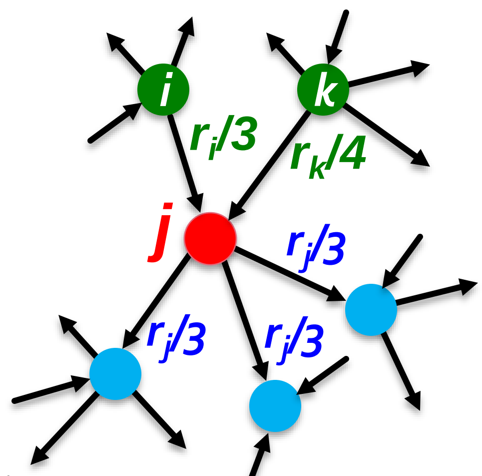
Example: PageRank Scores



Simple Recursive Formulation

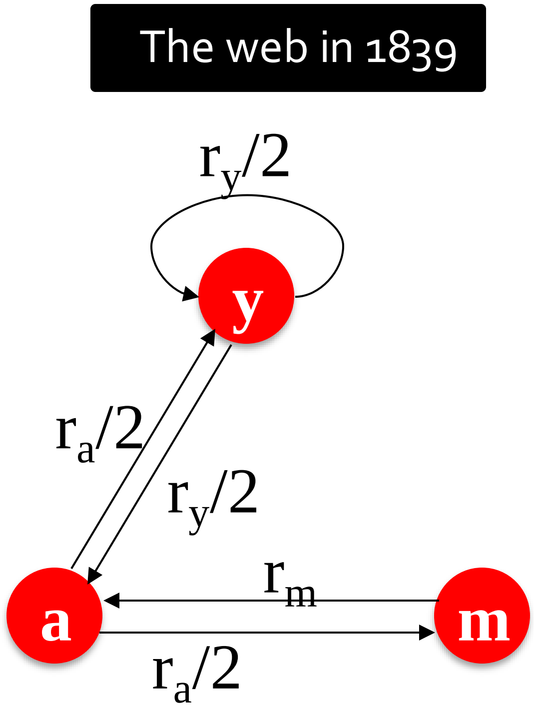
- Each link's vote is proportional to the **importance** of its source page
- If page j with importance r_j has n out-links, each link gets r_j / n votes
- Page j's own importance is the sum of the votes on its in-links

$$r_j = r_i/3 + r_k/4$$



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 page j



$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_i}$$

d_i … out-degree of node i

"Flow" equations:

$$\mathbf{r}_y = \mathbf{r}_y/2 + \mathbf{r}_a/2$$
$$\mathbf{r}_a = \mathbf{r}_y/2 + \mathbf{r}_m$$
$$\mathbf{r}_m = \mathbf{r}_a/2$$

r_j are the solutions to the "flow" equation

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 the scale factor
- Additional constraint forces uniqueness:
  - $\boldsymbol{r_y + r_a + r_m = 1}$
  - Solution: $\boldsymbol{r_y = \frac{2}{5},\ r_a = \frac{2}{5},\ r_m = \frac{1}{5}}$
- Gaussian elimination method works for small examples, but we need a better method for large web-size graphs
- We need a new formulation!

PageRank: Matrix Formulation

- **Stochastic adjacency matrix $\boldsymbol{M}$**
  - Let page i has d_i out-links
  - If $i \to j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$
    - $\boldsymbol{M}$ is a **column stochastic matrix**
      - Columns sum to 1
- **Rank vector $\boldsymbol{r}$:** 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**

$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d_i}}$$

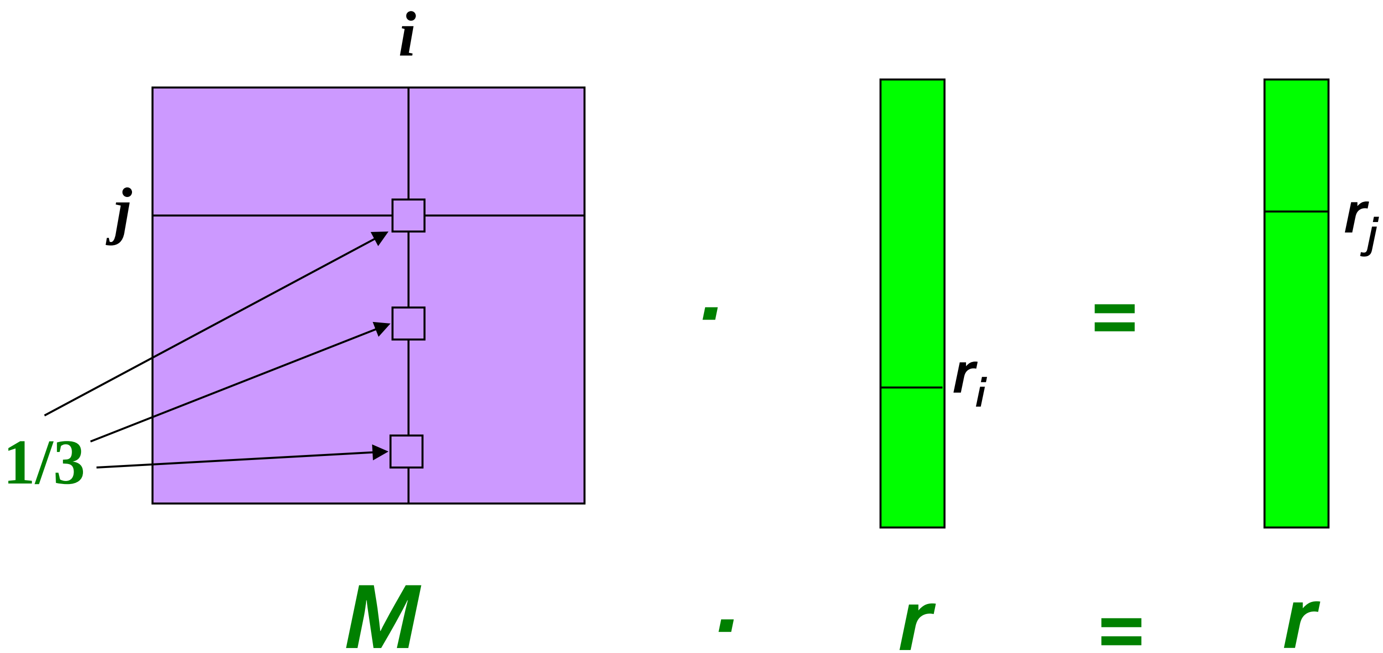
$$\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

Example

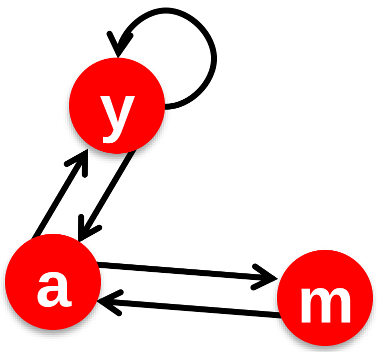
- Remember the flow equation: $r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d_i}}$
- Flow equation in the matrix form

$$M \cdot r = r$$

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



Example: Flow Equations & M



$$
\begin{array}{c|ccc}
 & r_y & r_a & r_m \\
\hline
r_y & \frac{1}{2} & \frac{1}{2} & 0 \\
r_a & \frac{1}{2} & 0 & 1 \\
r_m & 0 & \frac{1}{2} & 0
\end{array}
$$

$$r = M \cdot r$$

$$
\begin{aligned}
r_y &= r_y/2 + r_a/2 \\
r_a &= r_y/2 + r_m \\
r_m &= r_a/2
\end{aligned}
$$

$$
\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}
$$

Eigenvector Formulation

- **The flow equations can be written**

$$r = M \cdot r$$

- So the **rank vector** r is an **eigenvector** of the stochastic web matrix M
  - In fact, its first or principal eigenvector, with corresponding eigenvalue *1*
    - Largest eigenvalue of M is **1** since M is column stochastic (with non-negative entries)
      - *We know* **r** *is unit length and each column of* **M** *sums to one, so* $\boldsymbol{Mr} \leq \mathbf{1}$

NOTE: x is an eigenvector with the corresponding eigenvalue **λ** if:
$\boldsymbol{Ax} = \boldsymbol{\lambda x}$

- **We can now efficiently solve for *r*!**
The method is called Power iteration

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 $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$

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

d_i …. out-degree of node i

$|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the L_1 norm
So that **r** is a distribution (sums to 1)

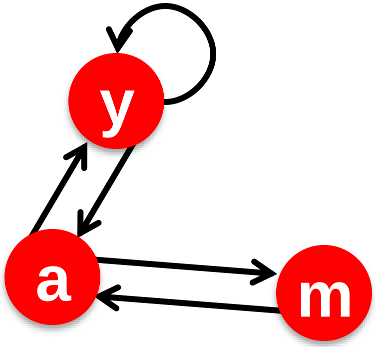
About 50 iterations is sufficient to estimate the limiting solution.

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**
- **Example:**

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 \\ 1/3 \\ 1/3 \end{matrix}$$

Iteration 0, 1, 2, …



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

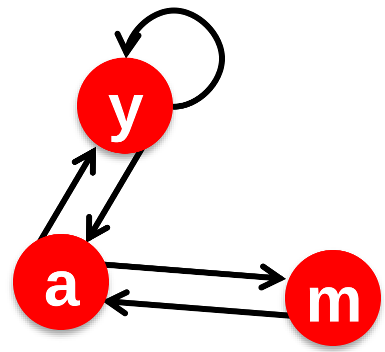
$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

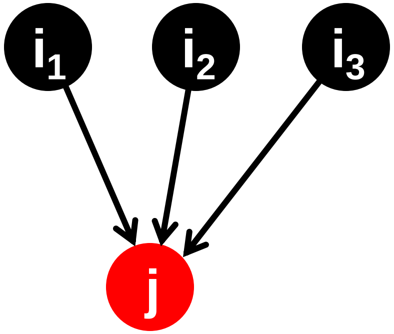
$\mathbf{r_y} = \mathbf{r_y}/2 + \mathbf{r_a}/2$
$\mathbf{r_a} = \mathbf{r_y}/2 + \mathbf{r_m}$
$\mathbf{r_m} = \mathbf{r_a}/2$

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

Iteration 0, 1, 2, …

Random Walk Interpretation

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

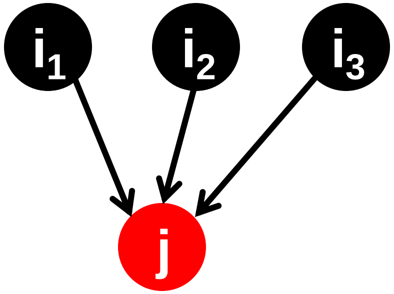


$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_{\mathrm{out}}(\mathrm{i})}$$

The Stationary Distribution

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

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



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

- Suppose the random walk reaches a state

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

  then $\boldsymbol{p}(\boldsymbol{t})$ is **stationary distribution** of a random walk
- **Our original rank vector** $\boldsymbol{r}$ satisfies $\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$
  - **So, $\boldsymbol{r}$ is a stationary distribution for the random walk**

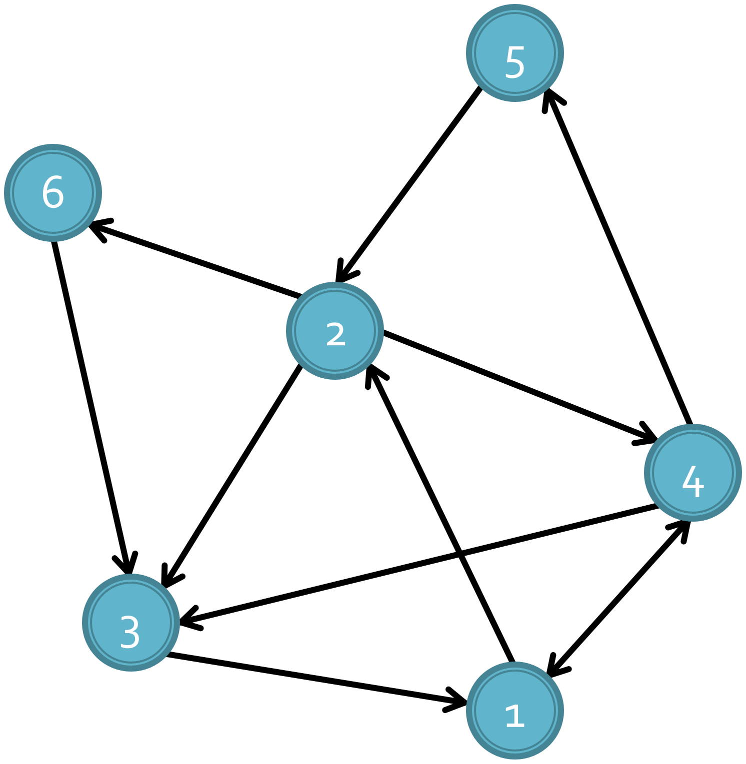
Existence and Uniqueness

- **A central result from the theory of random walks (a.k.a. Markov processes):**

For graphs that satisfy **certain conditions**, the **stationary distribution is unique** and eventually will be reached no matter what is the initial probability distribution at time **t = 0**

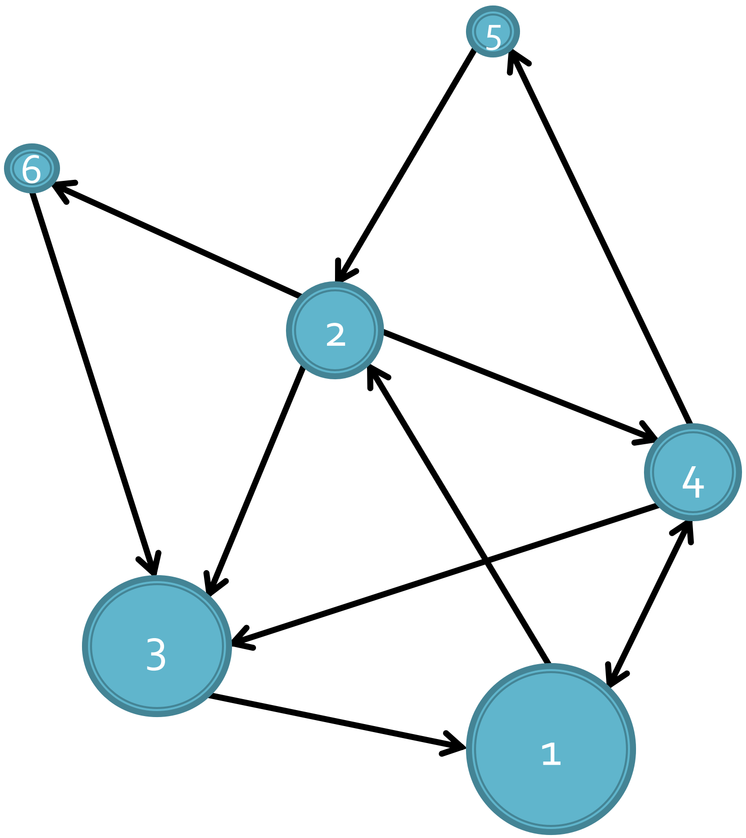
PageRank: test your intuition 1

- Which node has highest PageRank? Second highest?



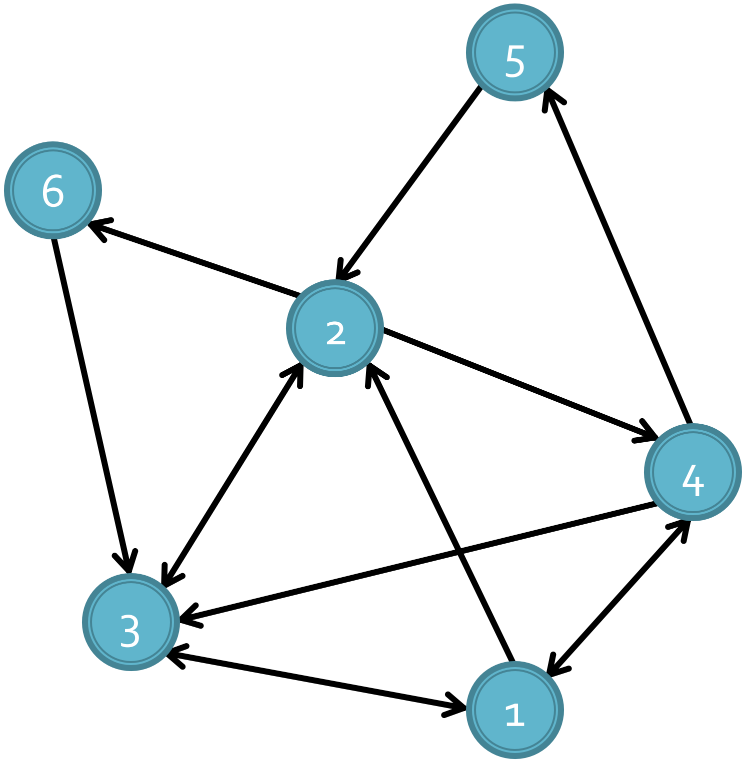
PageRank: test your intuition 1

- Node 1 has the highest PR, followed by Node 3
- Degree ≠ PageRank



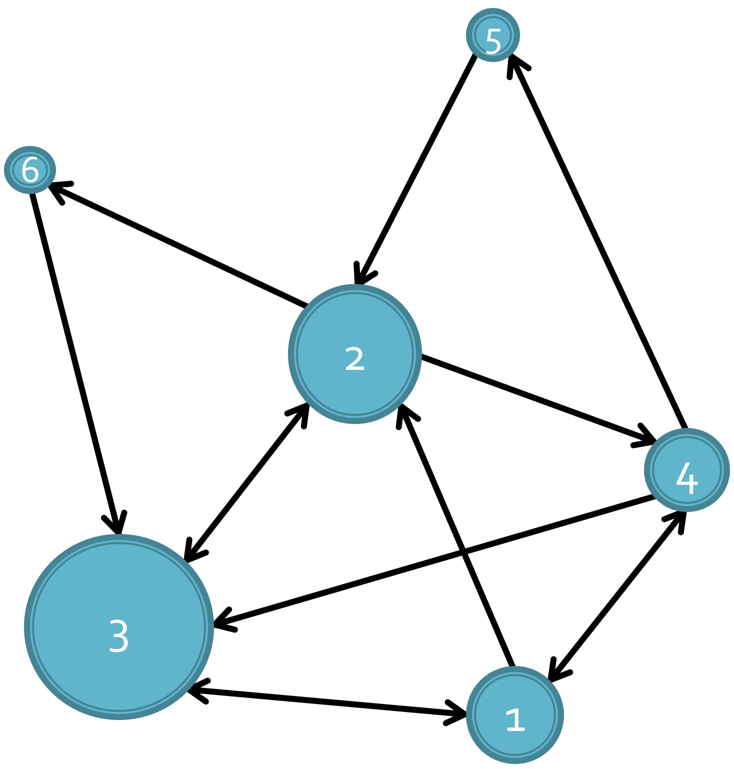
PageRank: test your intuition 2

- Add edge 3 -> 2, 1 -> 3. Now, which node has highest PageRank? Second highest?



PageRank: test your intuition 2

- Node 3 has the highest PR, followed by 2.
- Small changes to graph can change PR!



PageRank:
The Google Formulation

PageRank: Three Questions

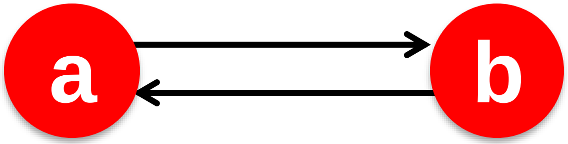
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_\mathrm{i}}$$

or equivalently

$$r = Mr$$

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

Does this converge?



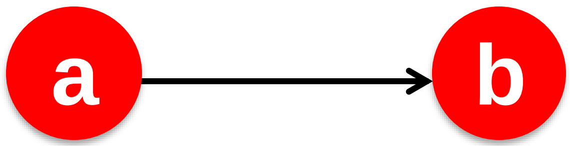
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_\mathrm{i}}$$

- **Example:**

$$\begin{array}{c} r_a \\ r_b \end{array} = \begin{array}{cccc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}$$

Iteration 0, 1, 2, …

Does it converge to what we want?



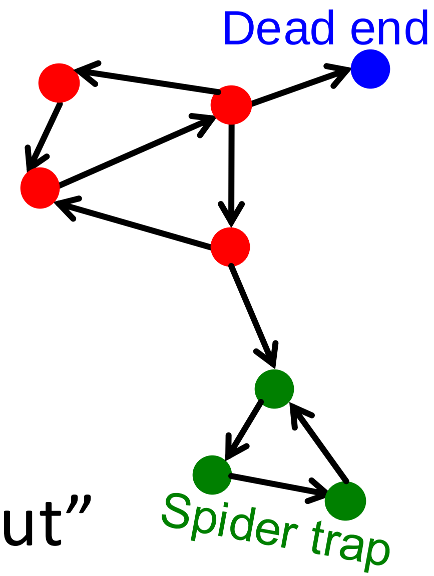
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_i}$$

- **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, …

PageRank: Problems



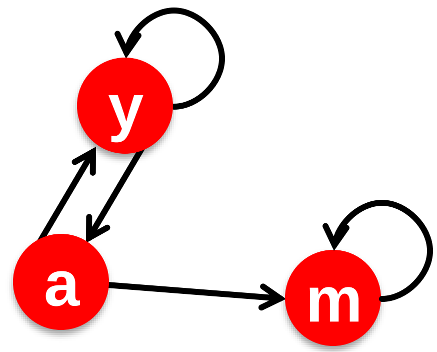
Two problems:

- **(1) Dead ends:** Some pages have no out-links
  - Random walk has "nowhere" to go to
  - Such pages cause importance to "leak out"

- **(2) Spider traps:** (all out-links are within the group)
  - Random walk gets "stuck" in a trap
  - And eventually spider traps absorb all importance

Problem: Spider Traps

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



m is a spider trap

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

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

- **Example:**

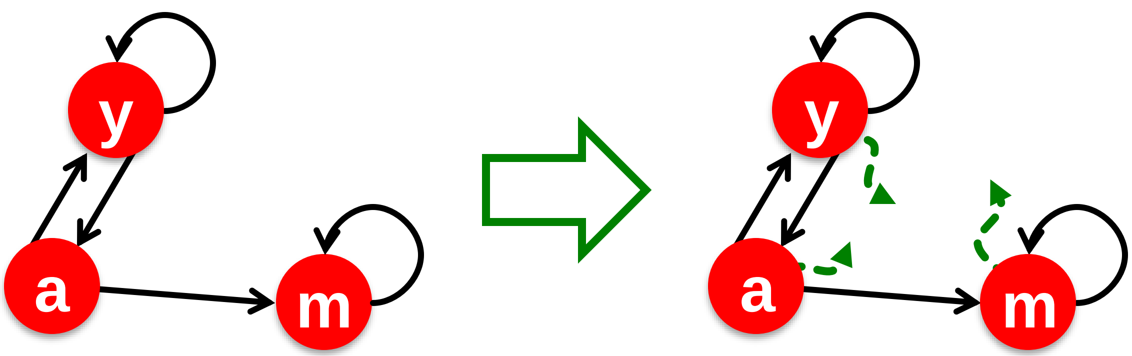
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

Iteration 0, 1, 2, …

All the PageRank score gets "trapped" in node m.

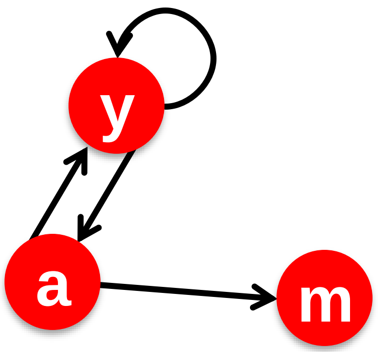
Solution: Teleports!

- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - β is typically 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 $r_j = 1/N$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

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

$\mathbf{r_a = r_y/2}$

$\mathbf{r_m = r_a/2}$

- **Example:**

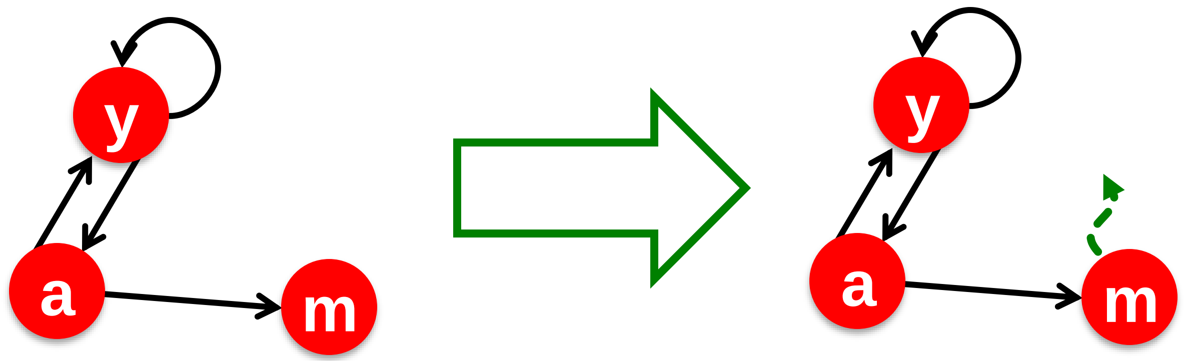
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

Iteration 0, 1, 2, …

Here the PageRank score “leaks” out since the matrix is not stochastic.

Solution: Always Teleport!

- **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?

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

- **Spider-traps** are not a problem, but with traps PageRank scores are **not** what we want
  - **Solution:** Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- **Dead-ends** are a problem
  - The matrix is not column stochastic so our initial assumptions are not met
  - **Solution:** Make matrix column stochastic by always teleporting when there is nowhere else to go

Solution: Random Teleports

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

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

d_i … out-degree of node i

This formulation assumes that M has no dead ends. We can either preprocess matrix M to remove all dead ends or explicitly follow random teleport links with probability 1.0 from dead-ends.

The Google Matrix

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

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

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

[1/N]$_{N\times N}$…N by N matrix where all entries are 1/N

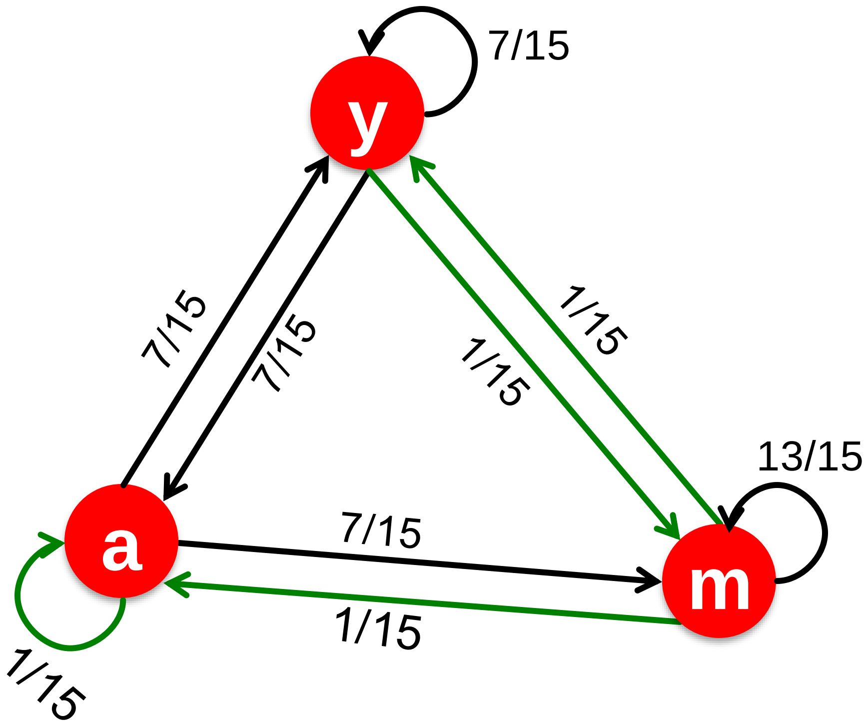
$$A = \beta\, M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$$

- **We have a recursive problem:** $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}$

  And the Power method still works!

- **What is β?**
  - In practice β =*0.8,0.9* (jump every *5* steps on avg.)

Random Teleports ($\beta = 0.8$)



$$
0.8 \underbrace{\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix}}_{\mathbf{M}} + 0.2 \underbrace{\begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}}_{[1/N]_{NxN}}
$$

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

y		1/3	0.33	0.24	0.26		7/33
a	=	1/3	0.20	0.20	0.18	. . .	5/33
m		1/3	0.46	0.52	0.56		21/33

How do we actually compute the PageRank?

Computing PageRank

- **Key step is matrix-vector multiplication**
  - $\boldsymbol{r}^{\text{new}} = \boldsymbol{A} \cdot \boldsymbol{r}^{\text{old}}$
- Easy if we have enough main memory to hold **A**, $\mathbf{r}^{\text{old}}$, $\mathbf{r}^{\text{new}}$
- **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

- $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}$, **where** $\boldsymbol{A_{ji} = \beta\, M_{ji} + \frac{1-\beta}{N}}$
- $r_j = \sum_{\mathrm{i}=1}^{N} A_{ji} \cdot r_i$
- $r_j = \sum_{i=1}^{N} \left[\beta\, M_{ji} + \frac{1-\beta}{N}\right] \cdot r_i$

$$= \sum_{\mathrm{i}=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{\mathrm{i}=1}^{N} r_i$$

$$= \sum_{\mathrm{i}=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N}$$

since $\sum r_i = 1$

- **So we get:** $\boldsymbol{r = \beta\, M \cdot r} + \left[\boldsymbol{\frac{1-\beta}{N}}\right]_{\boldsymbol{N}}$

Note: Here we assume **M** has no dead-ends

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

Sparse Matrix Formulation

- We just rearranged the **PageRank equation**

$$r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$$

  - where $[(1-\beta)/N]_N$ is a vector with all N entries $(1-\beta)/N$

- ***M*** is a **sparse matrix!** (with no dead-ends)
  - 10 links per node, approx $10N$ entries
- So in each iteration, we need to:
  - Compute $r^{new} = \beta M \cdot r^{old}$
  - Add a constant value $(1-\beta)/N$ to each entry in r^{new}
    - **Note if M contains dead-ends then $\sum_j r_j^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1**

PageRank: The Complete Algorithm

- **Input:** **Graph G and parameter β**
  - Directed graph G (can have **spider traps** and **dead ends**)
  - Parameter β
- **Output:** **PageRank vector r^{new}**
  - **Set:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| < \varepsilon$
    - $\forall j$: $\boldsymbol{r'^{new}_j = \sum_{i \to j} \beta \frac{r_i^{old}}{d_i}}$
      $\boldsymbol{r'^{new}_j = 0}$ if in-degree of $\boldsymbol{j}$ is **0**
    - **Now re-insert the leaked PageRank:**
      $\forall \boldsymbol{j}$: $\boldsymbol{r_j^{new} = r'^{new}_j + \frac{1-S}{N}}$ **where:** $S = \sum_j r'^{new}_j$
    - $\boldsymbol{r^{old} = r^{new}}$

If the graph has no dead-ends then the amount of leaked PageRank is **1-β**. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing **S**.

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**

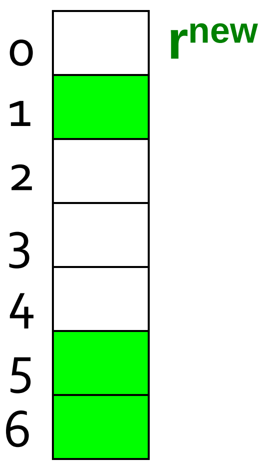
source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

Basic Algorithm: Update Step

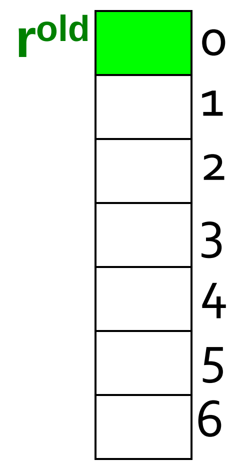
- **Assume enough RAM to fit r^{new} into memory**
  - Store r^{old} and matrix **M** on disk
- **1 step of power-iteration is:**

Initialize all entries of $\mathbf{r^{new}} = \mathbf{(1-\beta) / N}$
For each page i (of out-degree d_i):
  Read into memory: i, d_i, $dest_1, \ldots, dest_{d_i}$, $r^{old}(i)$
  For $\mathbf{j = 1 \ldots d_i}$
    $\mathbf{r^{new}(dest_j) \mathrel{+}= \beta\, r^{old}(i) / d_i}$

Assuming no dead ends



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

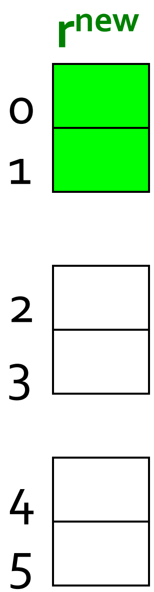


Analysis

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

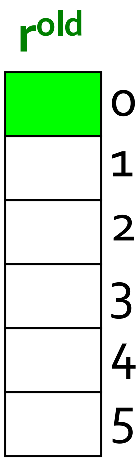
Extras

Block-based Update Algorithm



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

M

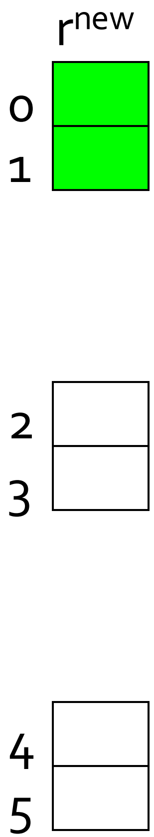


- Break r^{new} into k blocks that fit in memory
- Scan M and r^{old} once for each block

Analysis of Block Update

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

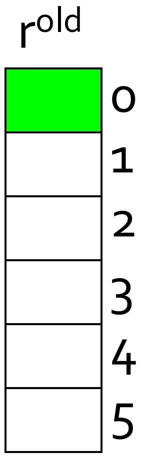
Block-Stripe Update Algorithm



src	degree	destination
0	4	0, 1
1	2	0

src	degree	destination
0	4	3
2	2	3

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



Break *M* into stripes! Each stripe contains only destination nodes in the corresponding block of r^{new}

Block-Stripe Analysis

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

where ε is a small number.

Some Problems with PageRank

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

Why Power Iteration works? (1)

- **Power iteration:**
  A method for finding dominant eigenvector (the vector corresponding to the largest eigenvalue)
  - $\boldsymbol{r}^{(1)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(0)}$
  - $\boldsymbol{r}^{(2)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(1)} = \boldsymbol{M}\left(\boldsymbol{M}\boldsymbol{r}^{(0)}\right) = \boldsymbol{M}^{2} \cdot \boldsymbol{r}^{(0)}$
  - $\boldsymbol{r}^{(3)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(2)} = \boldsymbol{M}\left(\boldsymbol{M}^{2}\boldsymbol{r}^{(0)}\right) = \boldsymbol{M}^{3} \cdot \boldsymbol{r}^{(0)}$
- **Claim:**
  Sequence $\boldsymbol{M} \cdot \boldsymbol{r}^{(0)}, \boldsymbol{M}^{2} \cdot \boldsymbol{r}^{(0)}, \ldots \boldsymbol{M}^{k} \cdot \boldsymbol{r}^{(0)}, \ldots$ approaches the dominant eigenvector of $\boldsymbol{M}$

Why Power Iteration works? (2)

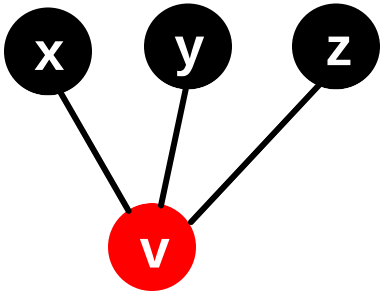
- **Claim:** Sequence $\boldsymbol{M} \cdot \boldsymbol{r}^{(0)}, \boldsymbol{M}^{2} \cdot \boldsymbol{r}^{(0)}, \ldots \boldsymbol{M}^{k} \cdot \boldsymbol{r}^{(0)}, \ldots$ approaches the dominant eigenvector of $\boldsymbol{M}$
- **Proof:**
  - Assume ***M*** has ***n*** linearly independent eigenvectors, $x_1, x_2, \ldots, x_n$ with corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, where $\lambda_1 > \lambda_2 > \cdots > \lambda_n$
  - Vectors $x_1, x_2, \ldots, x_n$ form a basis and thus we can write: $r^{(0)} = c_1\, x_1 + c_2\, x_2 + \cdots + c_n\, x_n$
  - $$\begin{aligned} \boldsymbol{M}\boldsymbol{r}^{(0)} &= \boldsymbol{M}(\boldsymbol{c_1}\, \boldsymbol{x_1} + \boldsymbol{c_2}\, \boldsymbol{x_2} + \cdots + \boldsymbol{c_n}\, \boldsymbol{x_n}) \\ &= c_1(Mx_1) + c_2(Mx_2) + \cdots + c_n(Mx_n) \\ &= c_1(\lambda_1 x_1) + c_2(\lambda_2 x_2) + \cdots + c_n(\lambda_n x_n) \end{aligned}$$
  - **Repeated multiplication on both sides produces** $$M^k r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \cdots + c_n(\lambda_n^k x_n)$$

Why Power Iteration works? (3)

- **Claim:** Sequence $\boldsymbol{M} \cdot \boldsymbol{r}^{(0)}, \boldsymbol{M}^{2} \cdot \boldsymbol{r}^{(0)}, \ldots \boldsymbol{M}^{k} \cdot \boldsymbol{r}^{(0)}, \ldots$ approaches the dominant eigenvector of $\boldsymbol{M}$
- **Proof (continued):**
  - Repeated multiplication on both sides produces
    $M^{k} r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \cdots + c_n(\lambda_n^k x_n)$
  - $M^{k} r^{(0)} = \lambda_1^k \left[c_1 x_1 + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \cdots + c_n \left(\frac{\lambda_n}{\lambda_1}\right)^k x_n \right]$
  - Since $\lambda_1 > \lambda_2$ then fractions $\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1} \ldots < 1$
    and so $\left(\frac{\lambda_i}{\lambda_1}\right)^k = 0$ as $k \to \infty$ (for all $i = 2 \ldots n$).
  - **Thus:** $\boldsymbol{M}^{k} \boldsymbol{r}^{(0)} \approx \boldsymbol{c_1}\left(\boldsymbol{\lambda_1^k x_1}\right)$
    - Note if $c_1 = 0$ then the method won't converge

PageRank for Undirected Graphs

- **Given an undirected graph with *N* nodes, where the nodes are pages and edges are hyperlinks**
- **Claim** [*Existence*]**:** For node v, $r_v = d_v/2m$ is a solution.



- Proof:
  - Iteration step: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$

$$r_v^{(t+1)} = \frac{r_x^t}{d_x} + \frac{r_y^t}{d_y} + \frac{r_z^t}{d_z}$$

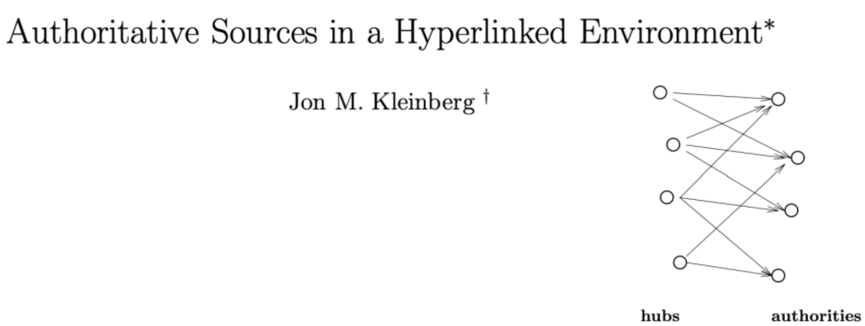
  - Substitute $r_i = d_i/2m$:

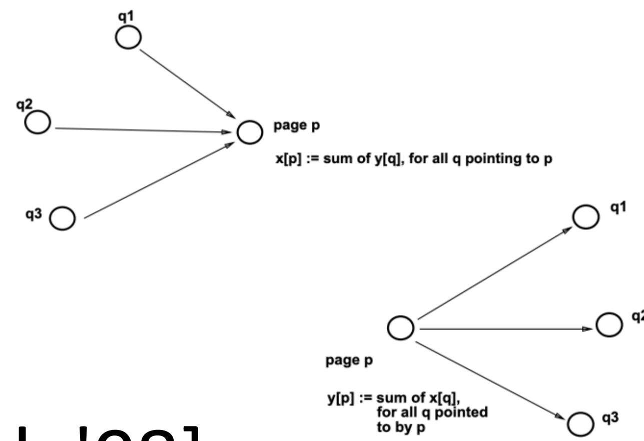
$$r_v^{(t+1)} = \frac{3}{2m}$$

- Done! Uniqueness: exercise! m = #edges

Historical note on Link Analysis

- Classic work: Markov chains, citation analysis
- RankDex patent [Robin Li, '96]
  - Key idea: use backlinks (led to Baidu!)
- HITS Algorithm [Kleinberg, SODA '98]
  - Key idea: iterative scoring!





- PageRank work [Page et al, '98]