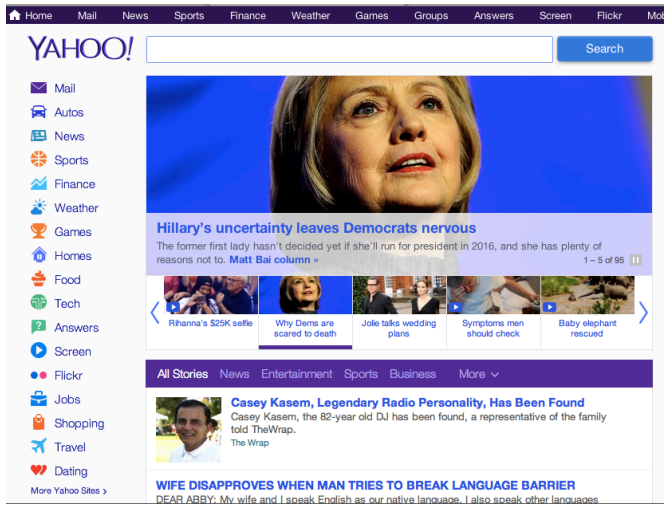
Link Mining

PageRank

From Stanford C246

Broad Question: How to organize the Web?

- **First try**: Human curated Web dictionaries
  - Yahoo,
  - DMOZ
  - LookSmart



- **Second try**: Web Search
  - Information Retrieval investigates documents in a small trusted set
  - Newspaper articles, Patents, etc

- **But**: Web is huge, full of untrusted documents, random things, web spam, etc.

Web Search Challenges

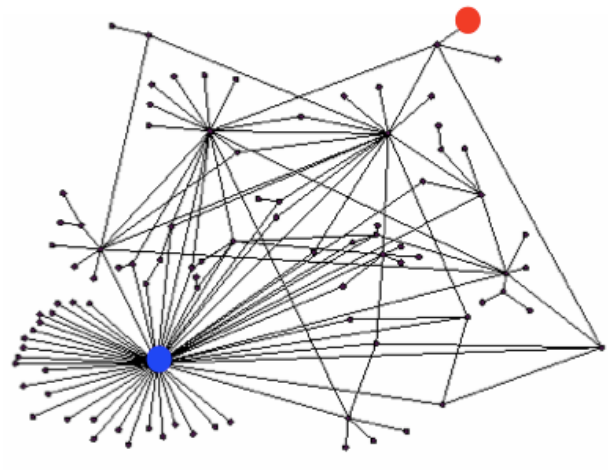
- Web contains many sources of information
  - Who to trust?
  - **Trick:** Trustworthy pages may point to each other

- 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"**

http://www.inf.unibz.it/~mkacimi/ vs www.stanford.edu

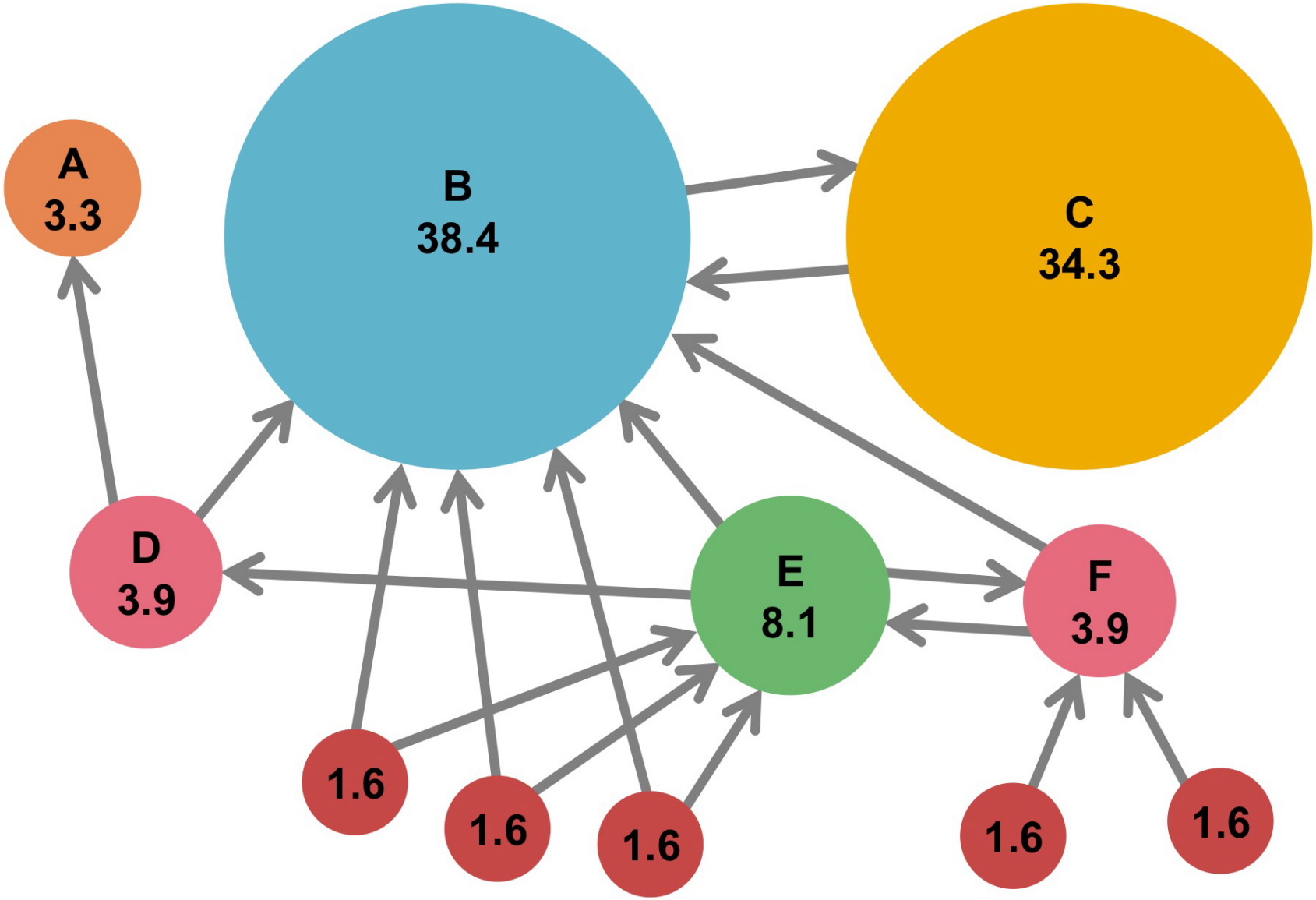


- There is large diversity in the web-graph node connectivity. So, what about ranking pages by link structure?

Idea: Links as Votes

- A page is more important if it has more links
  - In-links? Out-links?
- Think of links as votes:
  - www.stanford.edu **has 23,400 in-links**
  - http://www.inf.unibz.it/~mkacimi/ **has 1 in-link**
- Are all links equal?
  - Links from important pages count more
  - Recursive procedure

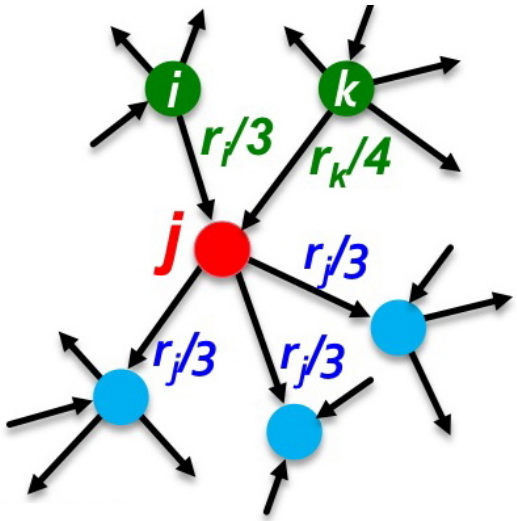
Example: PageRank Scores



Simple Recursive Formulation

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

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

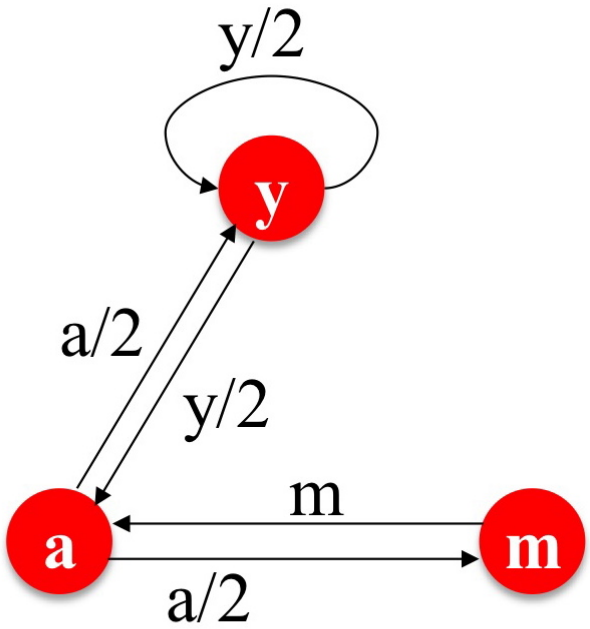


PageRank: The "Flow" Model

- 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" $\mathbf{r_j}$ for page **j** as

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

The web in 1839



"Flow" equations:

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

PageRank: Matrix Formulation

- **Stochastic adjacency matrix M**
  - Let page **i** has **d_i** outgoing links

$$M_{ij} = \begin{cases} \dfrac{1}{d_i} & \text{if} \quad i \rightarrow j \\ 0 & \text{else} \end{cases}$$

  - 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

$$r = M \,.\, r$$

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

Example

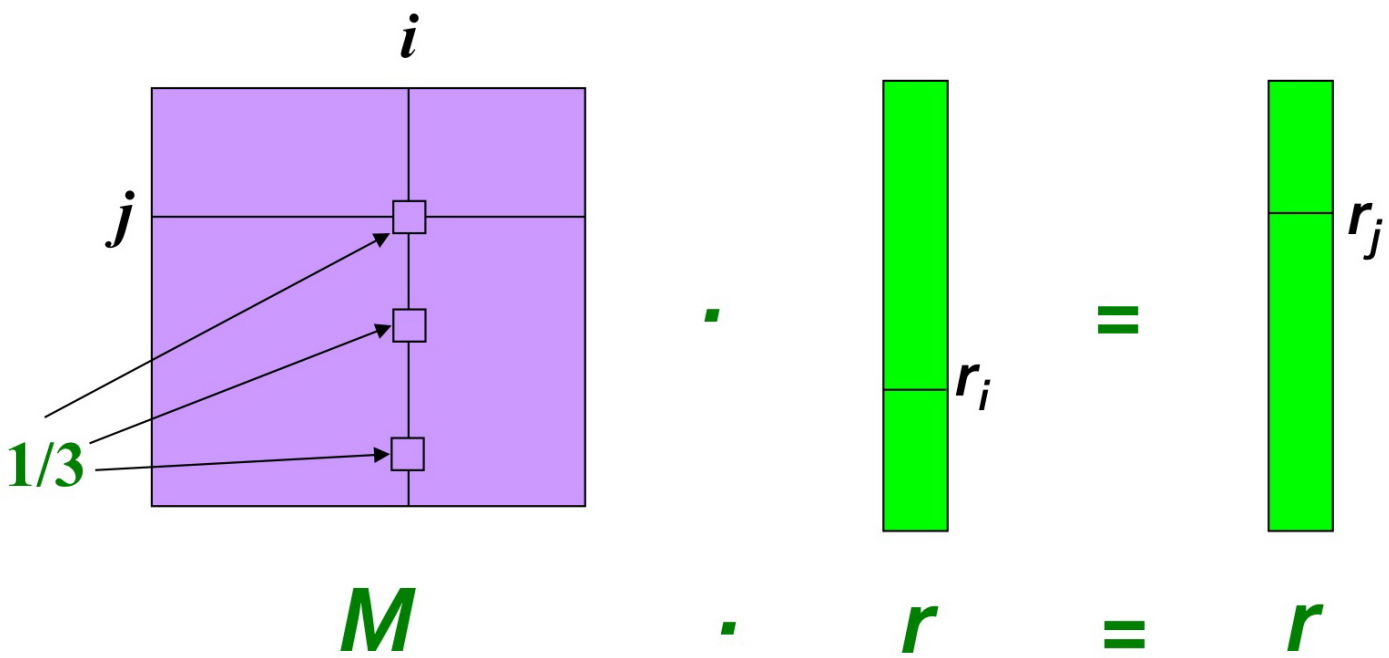
- Remember the flow equation:

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

- Flow equation in the matrix form

$$r = M \cdot r$$

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



Eigenvector Formulation

- The flow equations can be written

$$r = M \, . \, r$$

- So the rank vector **r** is an eigenvector of the stochastic web matrix **M**
- 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, **Mr<=1**
- 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: $r^{(0)} = \left[\frac{1}{N}, \ldots, \frac{1}{N}\right]^T$
  - Iterate $r^{(t+1)} = M.\, r^{(t)}$
  - Stop when $\left| r^{(t+1)} - r^{(t)} \right|_1 < \varepsilon$

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

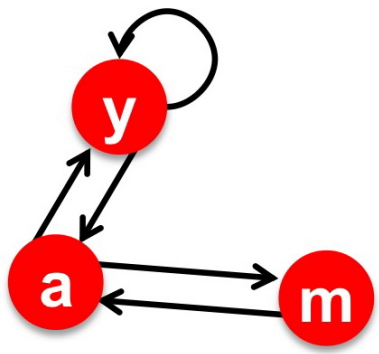
d_i : out degree of node i

$|X|_1 = \sum_{i=1}^{N} |X_i|$ is the L_1 norm

Can use any other vector norm. e.g., Euclidean

PageRank: How to solve?

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



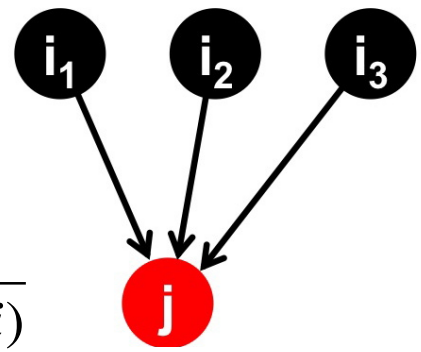
	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

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

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

Random Walk Interpretation

- Imagine a random web surfer:
  - At any time **t**, surfer is on some page **i**
  - At time **t+1**, the surfer follows an out-link from **i** uniformly at random
  - Ends up on some page **j** linked from **i**
  - Process repeats indefinitely



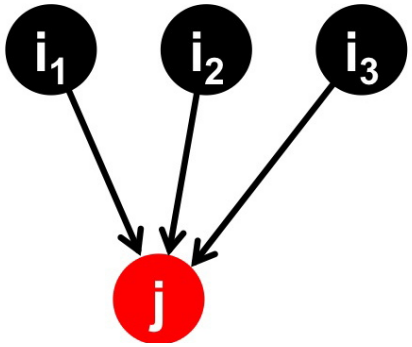
$$r_j = \sum_{i \to j} \frac{r_i}{d_{out}(i)}$$

- Let:
  - **P(t)**…vector whose **i**th coordinate is the probability that the surfer is at page **i** at time **t**
  - So, **p(t)** is a probability distribution over pages

Stationary Distribution

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

$$p(t+1) = M.p(t)$$



$$p(t+1) = M.p(t)$$

- Suppose the random walk reaches a state

$$p(t+1) = M.p(t) = p(t)$$

  - Then **p(t)** is stationary distribution of a random walk
- Our original rank vector **r** satisfies

$$r = M.r$$

  - So, **r** is a stationary distribution for the random walk

Existence and Uniqueness

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

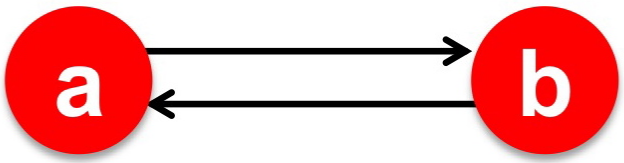
For graphs that satisfy certain conditions, the stationary distribution is unique and eventually will be reached no matter what the initial probability distribution at time t=0

PageRank: Three Questions

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_j} \quad \text{or equivalently} \quad \mathrm{r=M.r}$$

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

Does this converge?



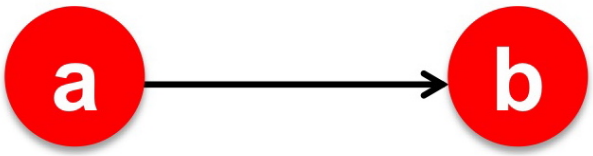
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_j}$$

- Example

$$\begin{pmatrix} r_a \\ r_b \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Iteration 0,1,2,

Does it converge to what we want?



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

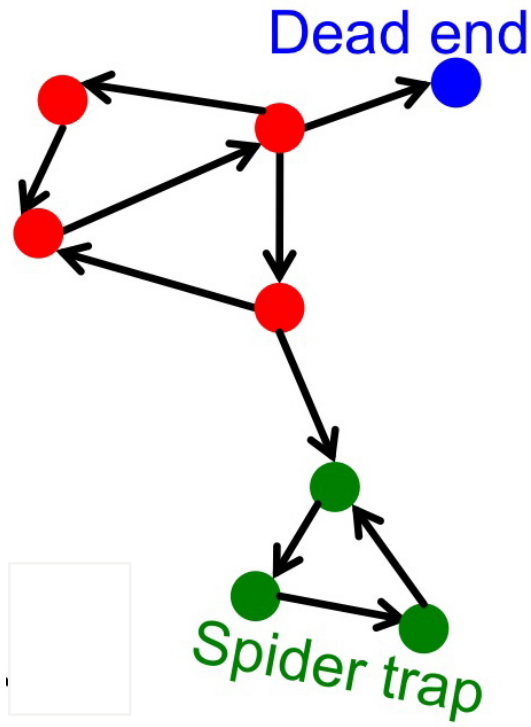
- Example

$$\begin{pmatrix} r_a \\ r_b \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Iteration 0,1,2,

PageRank: 2 Problems

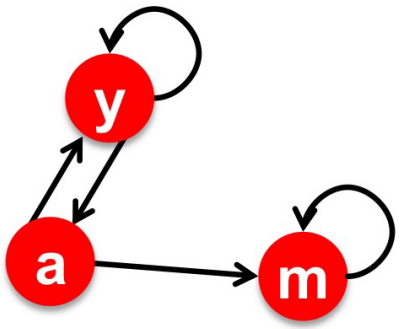
- **Dead Ends**
  - Some pages have no out-links
  - Random walk has "nowhere" to go to
  - Such pages cause importance to "leak out"



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

Problem1: Spider Traps

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



m is a spider trap

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

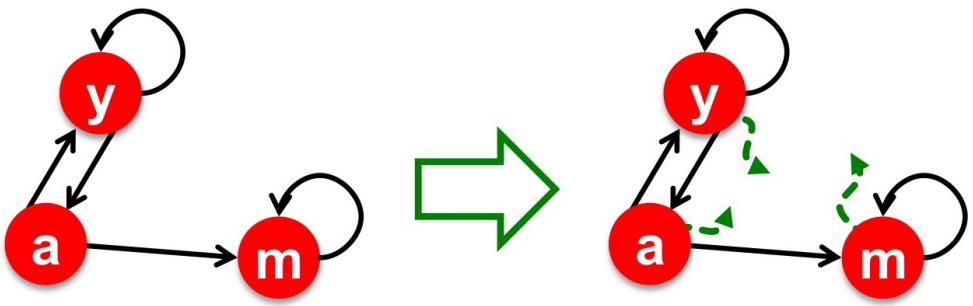
$$\mathbf{r_y = r_y/2 + r_a/2}$$
$$\mathbf{r_a = r_y/2}$$
$$\mathbf{r_m = r_a/2 + r_m}$$

- Example

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

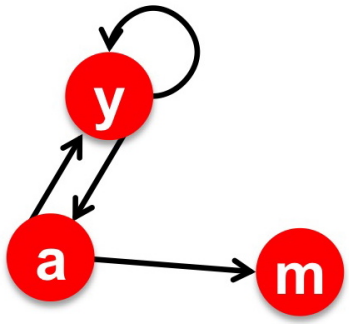
Solution: Teleports!

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



Problem2: Dead Ends

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



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

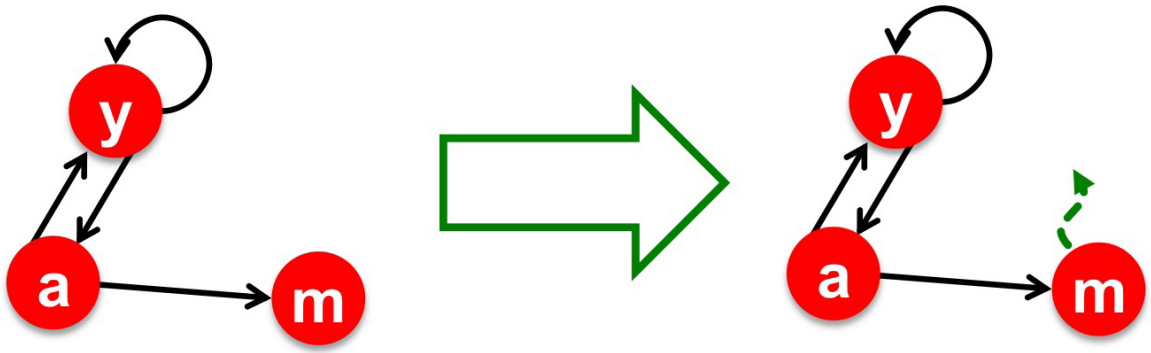
$$\mathbf{r_y = r_y/2 + r_a/2}$$
$$\mathbf{r_a = r_y/2}$$
$$\mathbf{r_m = r_a/2}$$

- Example

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

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 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 out 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 $1-\beta$, jump to some random page
- **PageRank equation**

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

d_i : out degree of node i

The Google Matrix

- PageRank equation

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

- **The Google Matrix A**

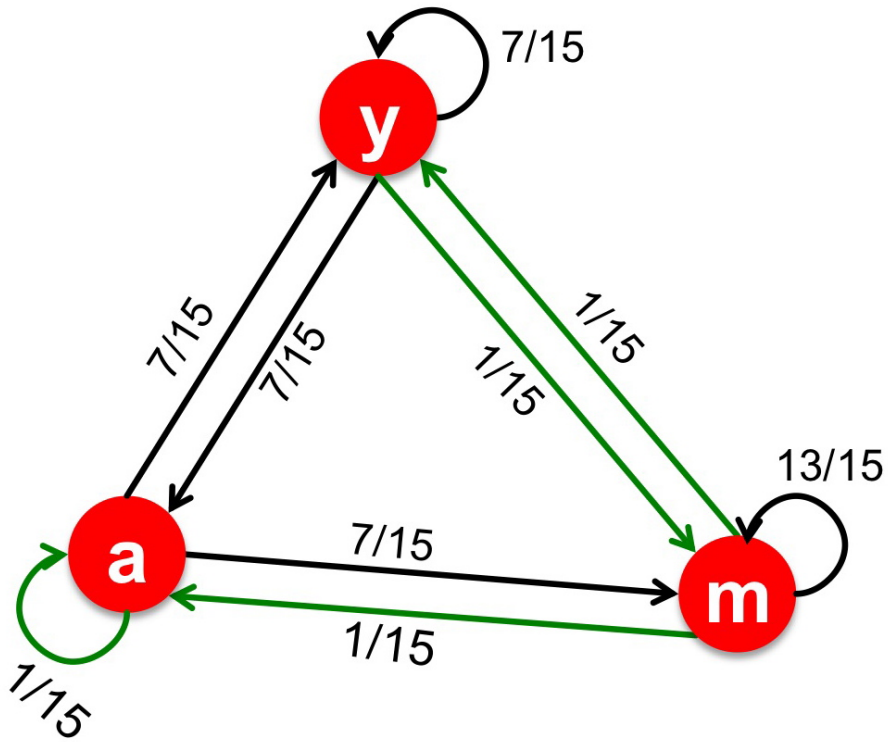
$$A = \beta . M + (1-\beta)\left[\frac{1}{N}\right]_{N \times N}$$

- We have a recursive problem $r = A.\ r$

  And the power method still works!

- What is β ?
  - In practice $\beta = 0.8, 0.9$ (make 5 steps on avg.,jump)

Random Teleports (β =0.8)



M **$[1/N]_{NxN}$**

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

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

A

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} \begin{pmatrix} 0.33 \\ 0.20 \\ 0.46 \end{pmatrix} \begin{pmatrix} 0.24 \\ 0.20 \\ 0.52 \end{pmatrix} \begin{pmatrix} 0.26 \\ 0.18 \\ 0.56 \end{pmatrix} \ldots \begin{pmatrix} 1/33 \\ 5/33 \\ 21/33 \end{pmatrix}$$

Computing PageRank

- Key step is matrix-vector multiplication

$$r^{new} = A.\ r^{old}$$

- 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!

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

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

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

Matrix Formulation

- Suppose there are **N** pages
- Consider page **i**, with $\mathbf{d_i}$ out-links
- We have

$$M_{ij} = \begin{cases} \dfrac{1}{d_i} & \text{if} \quad i \rightarrow j \\ 0 & \text{else} \end{cases}$$

- The random teleport is equivalent to:
  - Adding a teleport link from **i** to every other page and setting transition probability to $(1-\beta)/N$
  - Reducing the probability of following each out-link from $1/|d_i|$ to $\beta/|d_i|$
  - Equivalent: Tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

Rearranging the Equation

- 1: $r = A \cdot r$, where $A_{ij} = \beta M_{ji} + \dfrac{1-\beta}{N}$
- 2: $r_j = \sum_{i=1}^{N} A_{ij} \cdot r_i$
- 3: $r_j = \sum_{i=1}^{N} \left[\beta M_{ij} + \dfrac{1-\beta}{N} \right] \cdot r_i$

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

$$= \sum_{i=1}^{N} \beta M_{ij} \cdot r_i + \frac{1-\beta}{N} \quad \text{since} \sum_{i=1}^{N} r_i = 1$$

- So we get

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

Sparse Matrix Formulation

- We just rearranged the **PageRank** equation

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

- Where $\left[\frac{1-\beta}{N}\right]_N$ is a vector with all **N** entries **(1-β) /N**
- M is a **sparse matrix** (with no dead-ends)
  - 10 links per node, approximately 10.N entries
- So in each iteration, we need to:
  - Compute $r^{new} = \beta M . r^{old}$
  - Add a constant value **(1-β) /N** to each entry in $\mathbf{r^{new}}$
    - Note that if M contains dead-ends then $\sum_{j=1}^{N} r_j^{new} < 1$

    We also have to normalize $\mathbf{r^{new}}$ so that it sums to 1

PageRank: The Complete Algorithm

- **Input: Graph G and parameter β**
  - Directed graph $\boldsymbol{G}$ (can have **spider traps** and **dead ends**)
  - Parameter $\boldsymbol{\beta}$
- **Output: PageRank vector r^{new}**
  - **Set:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| > \varepsilon$
    - $\forall j$: $\boldsymbol{r'^{new}_j} = \sum_{i \to j} \boldsymbol{\beta} \frac{\boldsymbol{r_i^{old}}}{\boldsymbol{d_i}}$

      $\boldsymbol{r'^{new}_j} = \mathbf{0}$ if in-degree of $\boldsymbol{j}$ is **0**
    - **Now re-insert the leaked PageRank:**

      $\forall \boldsymbol{j}$: $\boldsymbol{r_j^{new}} = \boldsymbol{r'^{new}_j} + \frac{\mathbf{1}-\boldsymbol{S}}{\boldsymbol{N}}$ **where:** $S = \sum_j r'^{new}_j$
    - $\boldsymbol{r^{old}} = \boldsymbol{r^{new}}$