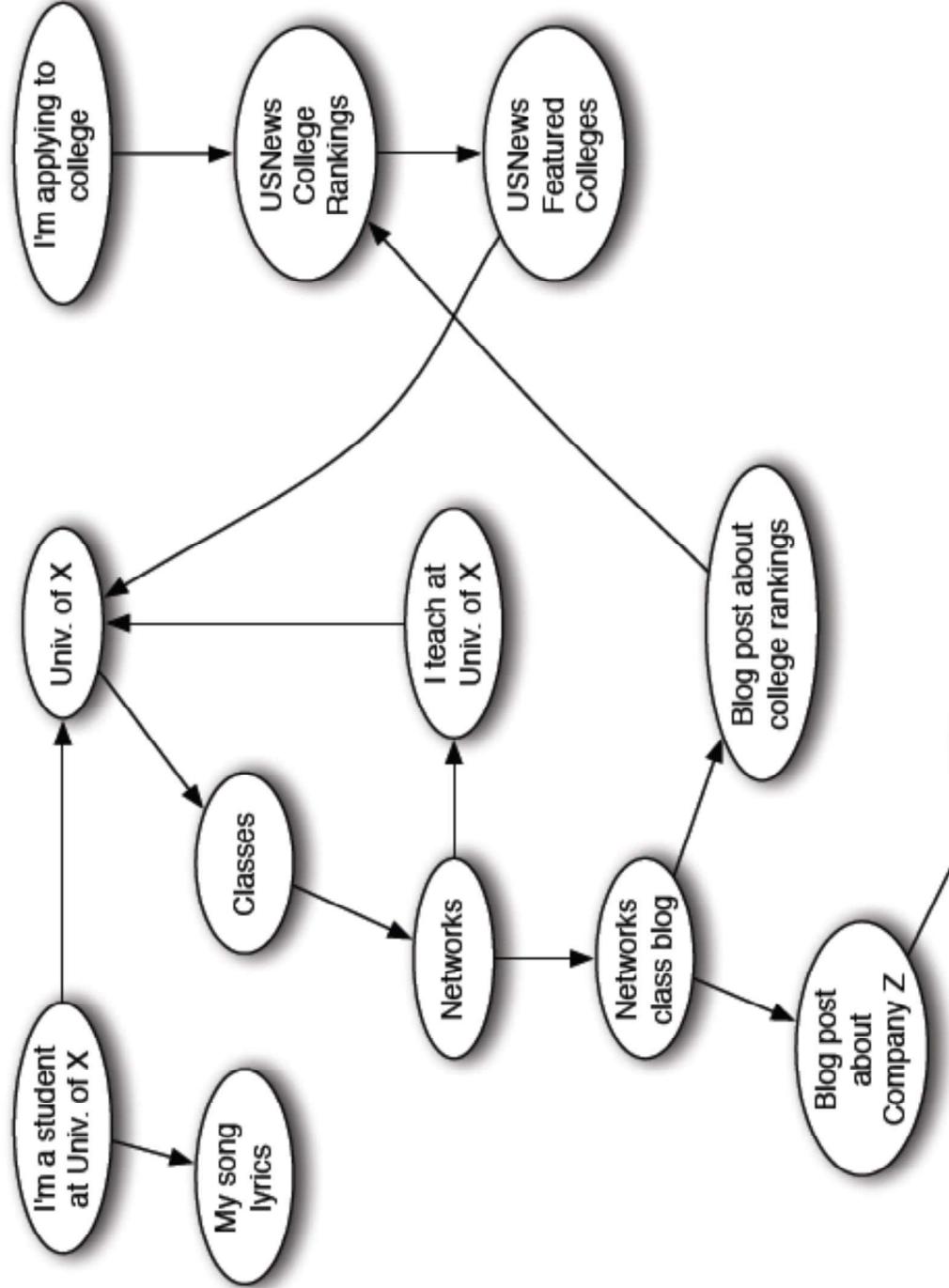


# Web as a Directed Graph



# 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.



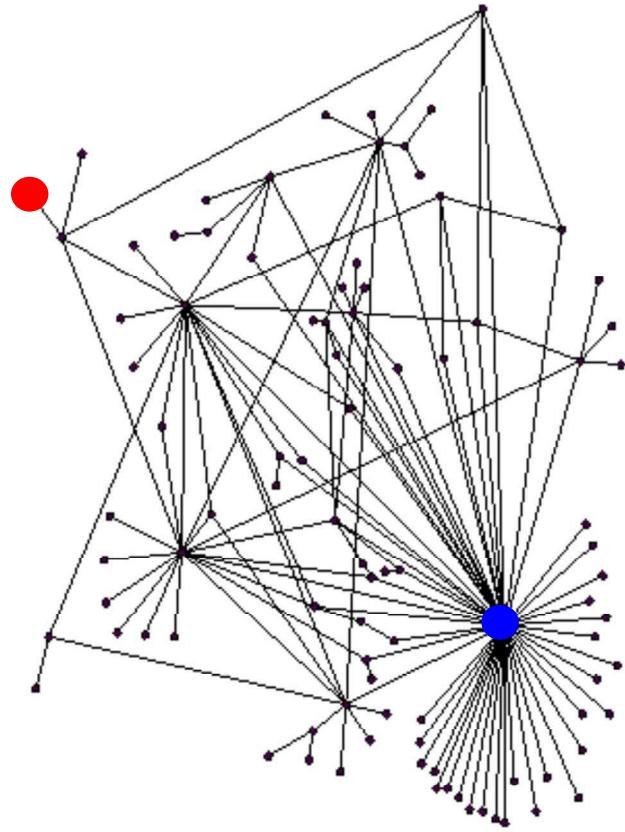
# Web Search: 2 Challenges

## 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”  
[www.pls.petdoge.com](http://www.pls.petdoge.com) vs. [www.stanford.edu](http://www.stanford.edu)



- There is large diversity in the web-graph node connectivity.

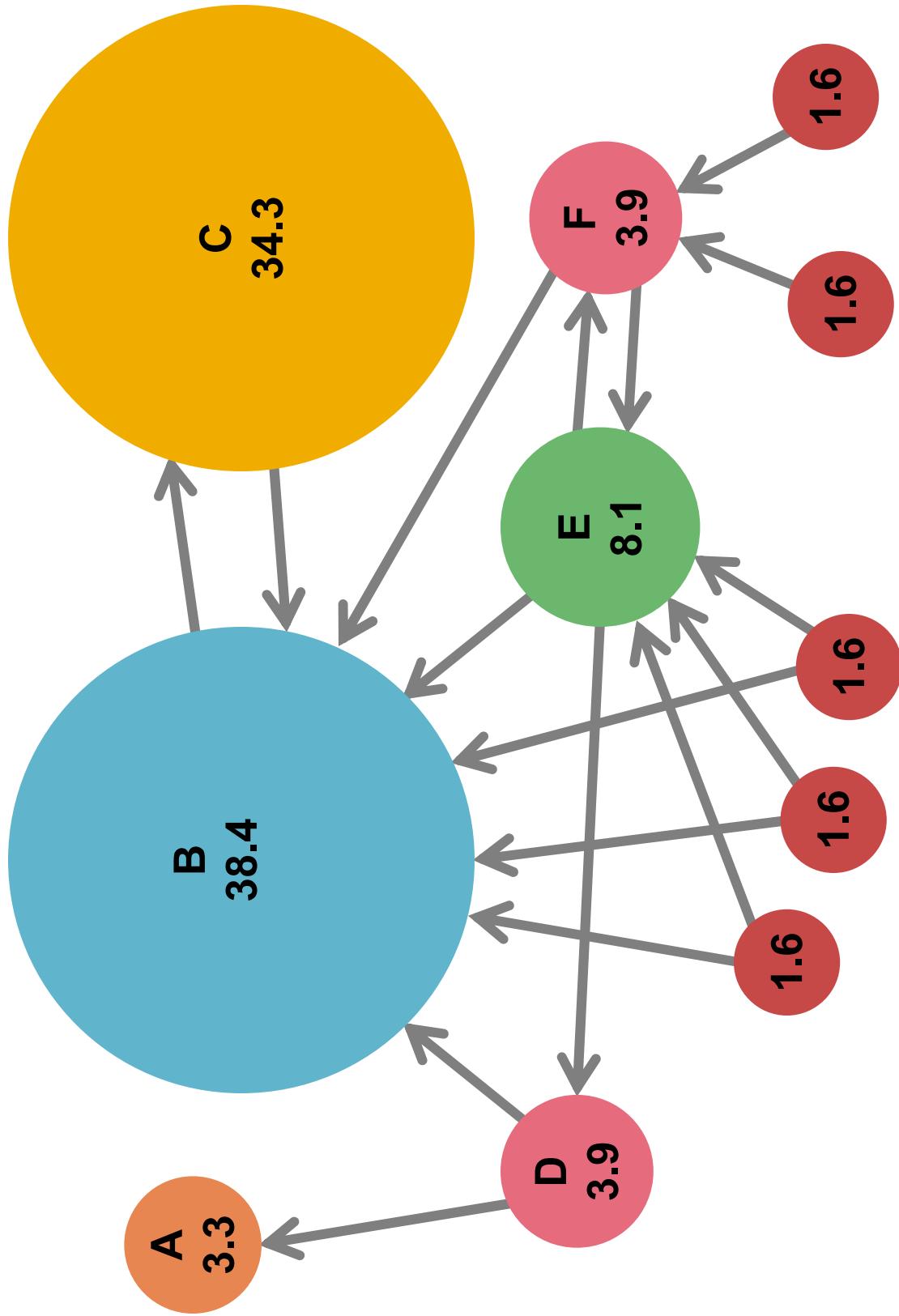
**Let's rank the pages by the link structure!**

# 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](http://www.stanford.edu) has 23,400 in-links
  - <http://pixelsfighting.com> has 1 in-link
- **Are all in-links are equal?**
- **Links from important pages count more**
  - Recursive question!

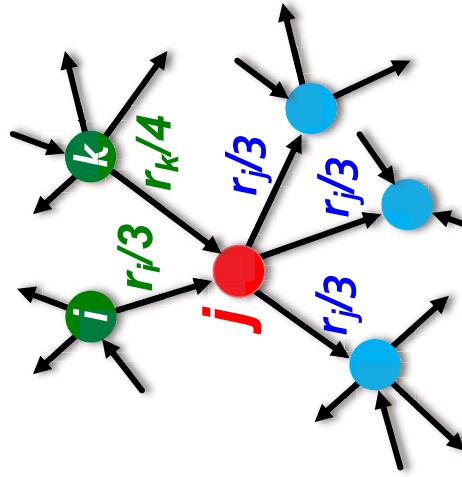
# Example: PageRank Scores



# 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

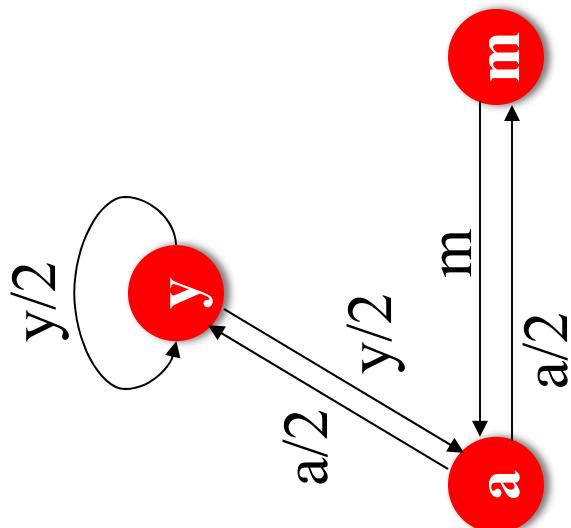
- Page  $j$ 's own importance is the sum of the votes on its in-links



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

# 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  $j$



The web in 1839

“Flow” equations:

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

$$r_a = r_y/2 + r_m$$

$$r_m = r_a/2$$

$d_i$  ... out-degree of node  $i$

# Solving the Flow Equations

- **3 equations, 3 unknowns,  
no constants**

- No unique solution

- All solutions equivalent modulo the scale factor

- **Additional constraint forces uniqueness:**

- $r_y + r_a + r_m = 1$

- **Solution:**  $r_y = \frac{2}{5}$ ,  $r_a = \frac{2}{5}$ ,  $r_m = \frac{1}{5}$

- **Gaussian elimination method works for  
small examples, but we need a better  
method for large web-size graphs**
- **We need a new formulation!**

Flow equations:  
 $r_y = r_y/2 + r_a/2$   
 $r_a = r_y/2 + r_m$   
 $r_m = r_a/2$

# PageRank: Matrix Formulation

## ■ Stochastic adjacency matrix $M$

- Let page  $i$  has  $d_i$  out-links

- If  $i \rightarrow j$ , then  $M_{ji} = \frac{1}{d_i}$  else  $M_{ji} = 0$

- $M$  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**

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

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

# Example

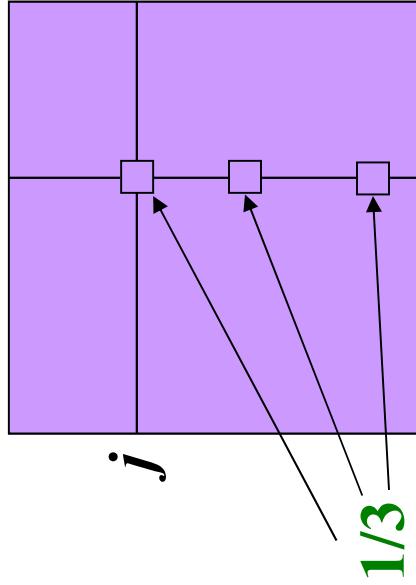
- Remember the flow equation:  $r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$
- Flow equation in the matrix form

$$M \cdot r = r$$

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

$i$

$j$



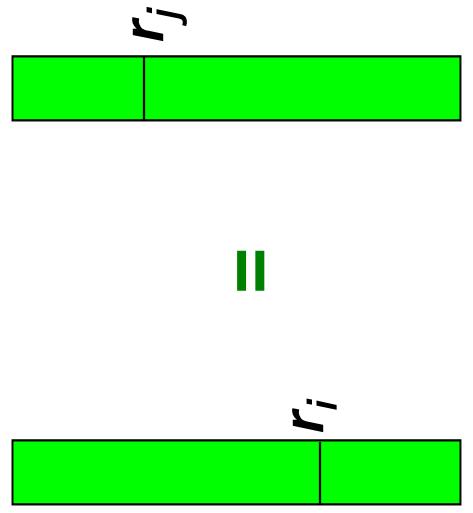
$M$

$\cdot$

$r$

$=$

$r$



# Eigenvector Formulation

- **The flow equations can be written**

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

- So the rank vector  $\mathbf{r}$  is an eigenvector of the stochastic web matrix  $\mathbf{M}$

- In fact, its first or principal eigenvector, with corresponding eigenvalue  $\mathbf{1}$

■ Largest eigenvalue of  $\mathbf{M}$  is  $\mathbf{1}$  since  $\mathbf{M}$  is column stochastic (with non-negative entries)

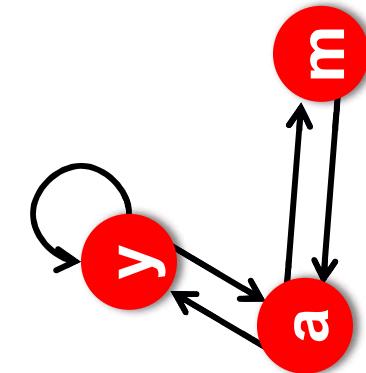
- We know  $\mathbf{r}$  is unit length and each column of  $\mathbf{M}$  sums to one, so  $\mathbf{M}\mathbf{r} \leq 1$

**NOTE:**  $\mathbf{x}$  is an eigenvector with the corresponding eigenvalue  $\lambda$  if:  
 $A\mathbf{x} = \lambda\mathbf{x}$

- **We can now efficiently solve for  $\mathbf{r}!$**

**The method is called Power iteration**

# Example: Flow Equations & M



	y	a	m
y	1/2	1/2	0
a	1/2	0	1
m	0	1/2	0

$$r = M \cdot r$$

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

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

$$r_a = r_y/2 + r_m$$

$$r_m = r_a/2$$

# 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:  $\mathbf{r}^{(0)} = [1/N, \dots, 1/N]^T$
    - Iterate:  $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
    - Stop when  $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \epsilon$
- $|\mathbf{x}|_1 = \sum_{1 \leq i \leq N} |x_i|$  is the L<sub>1</sub> norm
- Can use any other vector norm, e.g., Euclidean

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

# PageRank: How to solve?

## ■ Power Iteration:

- Set  $r_j = 1/N$
- $1: r'_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$

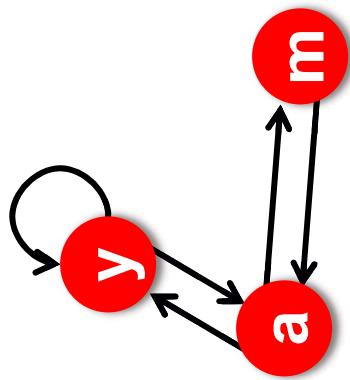
- $2: r = r'$

- Goto 1

## ■ Example:

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

Iteration 0, 1, 2, ...



	y	a	m
y	1/2	1/2	0
a	1/2	0	1
m	0	1/2	0

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

# PageRank: How to solve?

## ■ Power Iteration:

- Set  $r_j = 1/N$
- $1: r'_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$

- $2: r = r'$

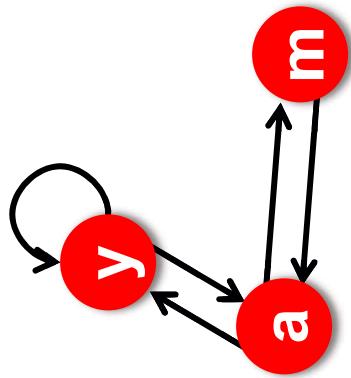
- Goto 1

## ■ Example:

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \quad \begin{bmatrix} 1/3 \\ 3/6 \\ 1/6 \end{bmatrix} \quad \begin{bmatrix} 5/12 \\ 1/3 \\ 3/12 \end{bmatrix} \quad \begin{bmatrix} 9/24 \\ 11/24 \\ 1/6 \end{bmatrix} \quad \dots$$

Iteration 0, 1, 2, ...

y	$\frac{1}{2}$	$\frac{1}{2}$	0
a	$\frac{1}{2}$	0	1
m	0	$\frac{1}{2}$	0



$$\begin{array}{c|ccccc} & y & a & m \\ \hline y & \frac{1}{2} & \frac{1}{2} & 0 \\ a & \frac{1}{2} & 0 & 1 \\ m & 0 & \frac{1}{2} & 0 \end{array}$$

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

# Why Power Iteration works? (I)

Details!

## ■ **Power iteration:**

A method for finding dominant eigenvector (the vector corresponding to the largest eigenvalue)

- $\mathbf{r}^{(1)} = M \cdot \mathbf{r}^{(0)}$
  - $\mathbf{r}^{(2)} = M \cdot \mathbf{r}^{(1)} = M(M\mathbf{r}^{(1)}) = M^2 \cdot \mathbf{r}^{(0)}$
  - $\mathbf{r}^{(3)} = M \cdot \mathbf{r}^{(2)} = M(M^2\mathbf{r}^{(0)}) = M^3 \cdot \mathbf{r}^{(0)}$
- ## ■ **Claim:**
- Sequence  $M \cdot \mathbf{r}^{(0)}, M^2 \cdot \mathbf{r}^{(0)}, \dots, M^k \cdot \mathbf{r}^{(0)}, \dots$  approaches the dominant eigenvector of  $M$

# Why Power Iteration works? (2)

Details!

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

# Why Power Iteration works? (3)

Details!

- **Claim:** Sequence  $\mathbf{M} \cdot \mathbf{r}^{(0)}, \mathbf{M}^2 \cdot \mathbf{r}^{(0)}, \dots, \mathbf{M}^k \cdot \mathbf{r}^{(0)}, \dots$  approaches the dominant eigenvector of  $\mathbf{M}$
- **Proof (continued):**

- Repeated multiplication on both sides produces

$$\mathbf{M}^k \mathbf{r}^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \dots + c_n(\lambda_n^k x_n)$$

$$\mathbf{M}^k \mathbf{r}^{(0)} = \lambda_1^k \left[ c_1 x_1 + c_2 \left( \frac{\lambda_2}{\lambda_1} \right)^k x_2 + \dots + c_n \left( \frac{\lambda_2}{\lambda_1} \right)^k x_n \right]$$

- Since  $\lambda_1 > \lambda_2$  then fractions  $\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1} \dots < 1$  and so  $\left( \frac{\lambda_i}{\lambda_1} \right)^k = 0$  as  $k \rightarrow \infty$  (for all  $i = 2 \dots n$ ).

- **Thus:**  $\mathbf{M}^k \mathbf{r}^{(0)} \approx c_1(\lambda_1^k x_1)$

- Note if  $c_1 = 0$  then the method won't converge

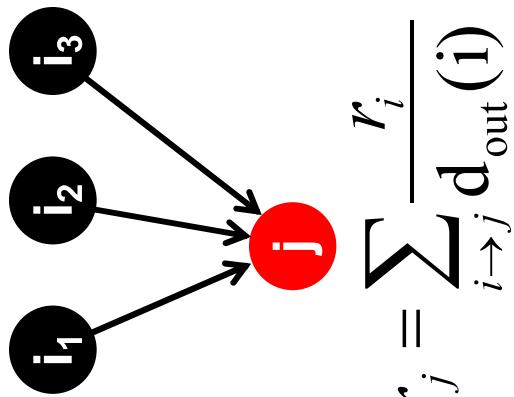
# Random Walk Interpretation

## ■ **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

## ■ **Let:**

- $p(t)$  ... vector whose  $i^{\text{th}}$  coordinate is the prob. that the surfer is at page  $i$  at time  $t$
- So,  $p(t)$  is a probability distribution over pages



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

# PageRank: The Google Formulation

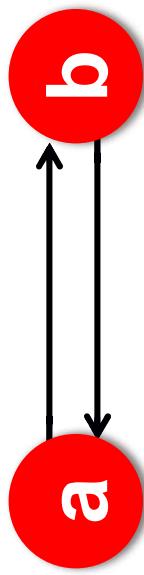
# PageRank: Three Questions

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

or  
equivalently       $r = Mr$

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

# Does this converge?



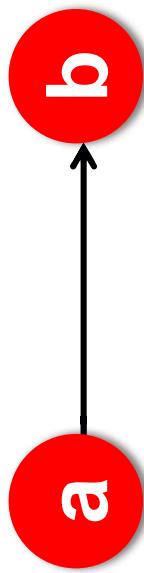
$$r_j^{(t+1)} = \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i}$$

- **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?



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

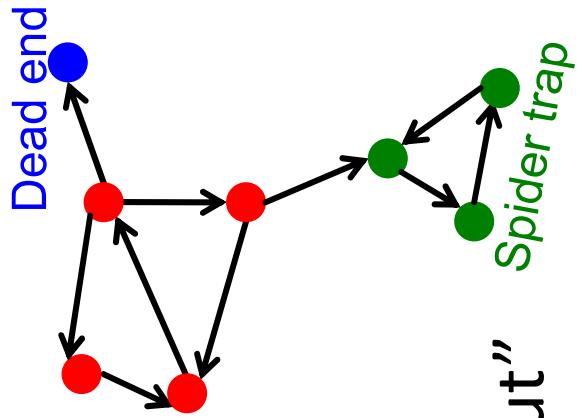
## ■ Example:

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

Iteration 0, 1, 2, ...

# PageRank: Problems

## 2 problems:



- (1) Some pages are **dead ends** (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 walked gets “stuck” in a trap
  - And eventually spider traps absorb all importance

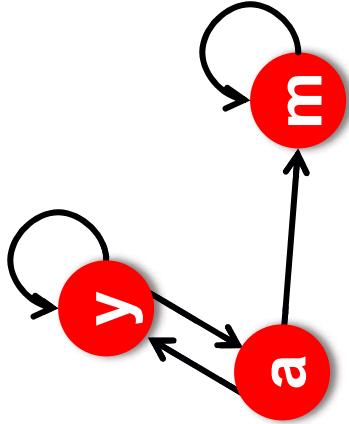
# Problem: Spider Traps

- **Power Iteration:**

- Set  $r_j = 1$

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

- And iterate



- m is a spider trap

y	$\frac{1}{2}$	$\frac{1}{2}$	0
a	$\frac{1}{2}$	0	0
m	0	$\frac{1}{2}$	1

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

- **Example:**

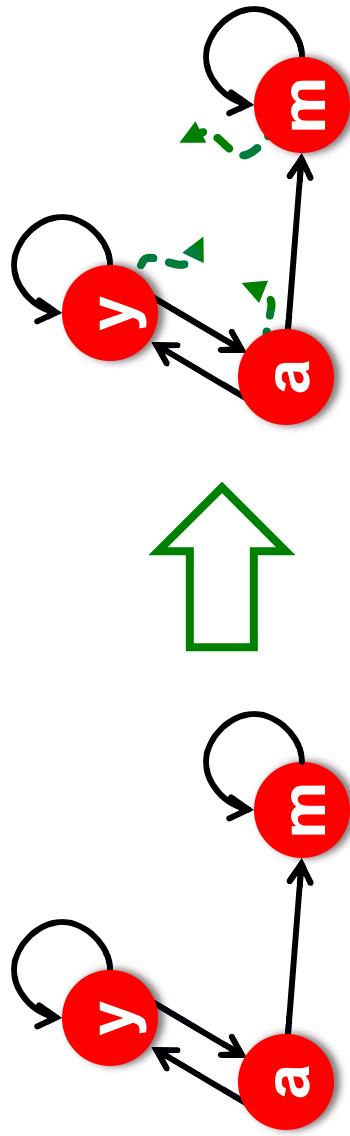
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \quad \begin{bmatrix} 2/6 \\ 1/6 \\ 3/6 \end{bmatrix} \quad \begin{bmatrix} 3/12 \\ 2/12 \\ 7/12 \end{bmatrix} \quad \begin{bmatrix} 5/24 \\ 3/24 \\ 16/24 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\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

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.  $\beta$ , follow a link at random
  - With prob.  $1-\beta$ , jump to some random page
- Common values for  $\beta$  are in the range 0.8 to 0.9
- Surfer will teleport out of spider trap within a few time steps



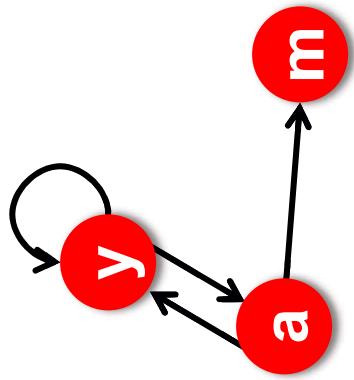
# Problem: Dead Ends

- **Power Iteration:**

- Set  $r_j = 1$

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

- And iterate



- **Example:**

	y	a	m
y	1/2	1/2	0
a	1/2	0	0
m	0	1/2	0

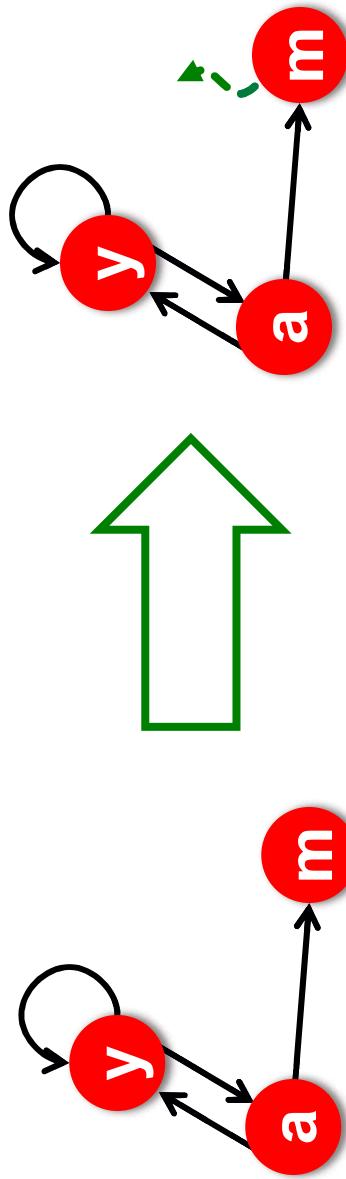
$$\begin{aligned}r_y &= r_y/2 + r_a/2 \\r_a &= r_y/2 \\r_m &= r_a/2\end{aligned}$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/3 & 2/6 & 3/12 & 5/24 \\ 1/3 & 1/6 & 2/12 & 3/24 \\ 1/3 & 1/6 & 1/12 & 2/24 \end{bmatrix} \quad \text{Iteration 0, 1, 2, ...}$$

Here the PageRank "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	$\frac{1}{2}$	$\frac{1}{2}$	0
a	$\frac{1}{2}$	0	0
m	0	$\frac{1}{2}$	$\frac{1}{2}$

# 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  $\beta$ , follow a link at random
- With probability  $1-\beta$ , jump to some random page

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

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

$d_i$  ... 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]

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

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

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

$$A = \beta M + (1 - \beta) \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{N \times N}$$

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

**And the Power method still works!**

- **What is  $\beta$ ?**

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

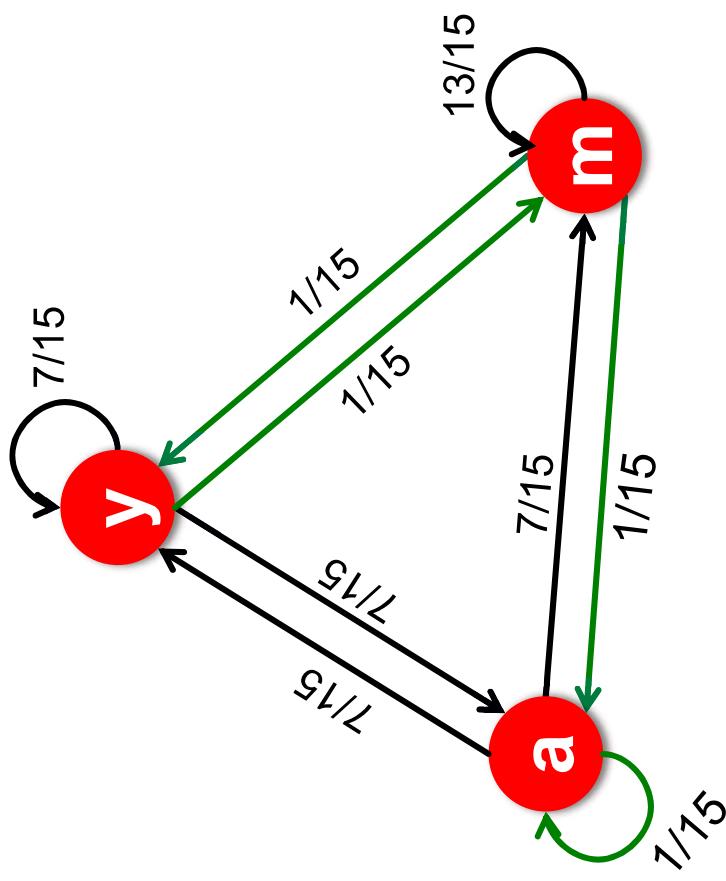
# Random Teleports ( $\beta = 0.8$ )

$[1/N]_{N \times N}$

$$\begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$M$

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



$A$

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

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

How do we actually compute  
the PageRank?

# Computing Page Rank

- **Key step is matrix-vector multiplication**

- $r^{\text{new}} = A \cdot r^{\text{old}}$
- Easy if we have enough main memory to hold  $A$ ,  $r^{\text{old}}$ ,  $r^{\text{new}}$
- **Say  $N = 1 \text{ billion pages}$** 
  - 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!

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

$$A = 0.8 \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 1 \end{bmatrix} + 0.2 \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

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

# Rearranging the Equation

- $\mathbf{r} = A \cdot \mathbf{r}$ , where  $A_{ji} = \beta M_{ji} + \frac{1-\beta}{N}$
- $r_j = \sum_{i=1}^N A_{ji} \cdot r_i$
- $r_j = \sum_{i=1}^N \left[ \beta M_{ji} + \frac{1-\beta}{N} \right] \cdot r_i$ 
$$= \sum_{i=1}^N \beta M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{i=1}^N r_i$$
$$= \sum_{i=1}^N \beta M_{ji} \cdot r_i + \frac{1-\beta}{N}$$
 since  $\sum r_i = 1$
- **So we get:  $\mathbf{r} = \beta M \cdot \mathbf{r} + \left[ \frac{1-\beta}{N} \right]_N$**

**Note:** Here we assumed  $\mathbf{M}$  has no dead-ends

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

# PageRank: The Complete Algorithm

- **Input:** Graph  $G$  and parameter  $\beta$ 
  - Directed graph  $G$  (can have spider traps and dead ends)
  - Parameter  $\beta$
- **Output:** PageRank vector  $r^{new}$

- Set:  $r_j^{old} = \frac{1}{N}$
- **repeat until convergence:**  $\sum_j |r_j^{new} - r_j^{old}| > \epsilon$ 
  - $\forall j: r'^{new}_j = \sum_{i \rightarrow j} \beta \frac{r_i^{old}}{d_i}$
  - $r'^{new}_j = 0$  if in-degree of  $j$  is 0
- Now re-insert the leaked PageRank:
  - $\forall j: r_j^{new} = r'^{new}_j + \frac{1-s}{N}$  where:  $S = \sum_j r'^{new}_j$
  - $r^{old} = r^{new}$

If the graph has no dead-ends then the amount of leaked PageRank is **1- $\beta$** . 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  $4 * 10 * 1 \text{ billion} = 40\text{GB}$
- **Still won't fit in memory, but will fit on disk**

**source node**    **degree**    **destination nodes**

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  $r^{new} = (1-\beta) / N$

For each page  $i$  (of out-degree  $d_i$ ):

Read into memory:  $i, d_i, dest_1, \dots, dest_{d_i}, r^{old}(i)$

For  $j = 1 \dots d_i$

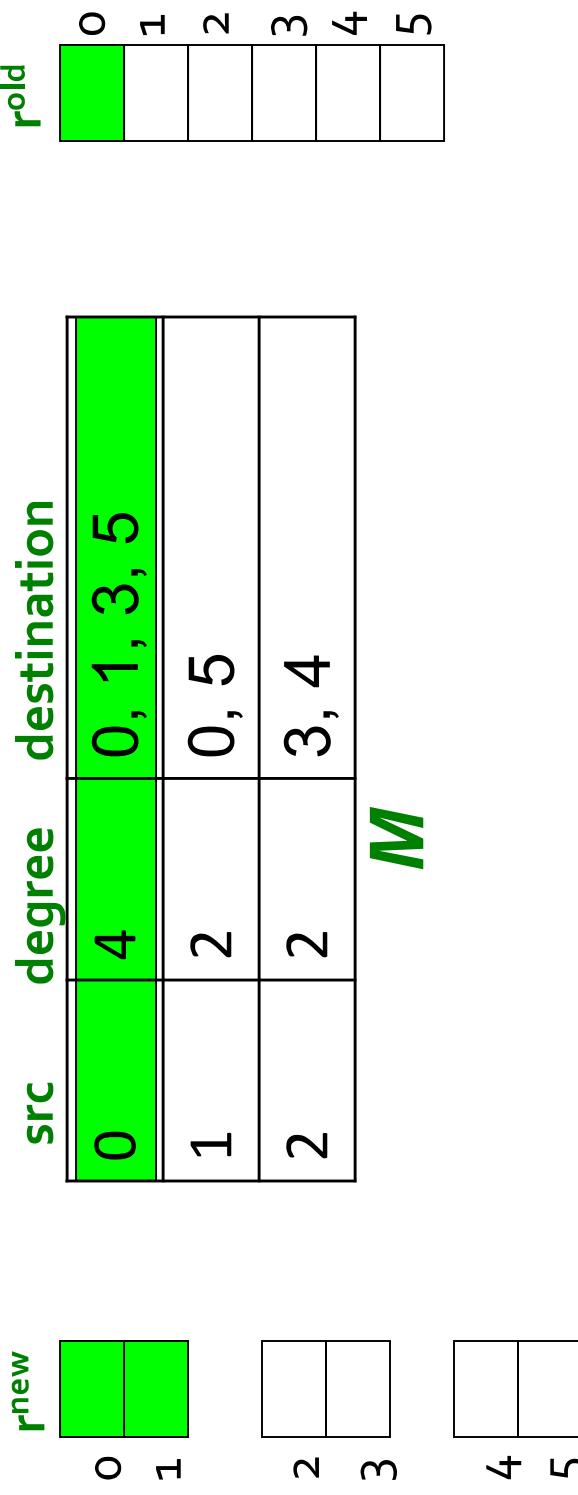
$r^{new}(dest_j) += \beta r^{old}(j) / d_i$

	$r^{new}$	$source$	$degree$	$destination$	$r^{old}$
0					0
1		3	1	5, 6	1
2					2
3					3
4					4
5					5
6					6

# 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?

# Block-based Update Algorithm

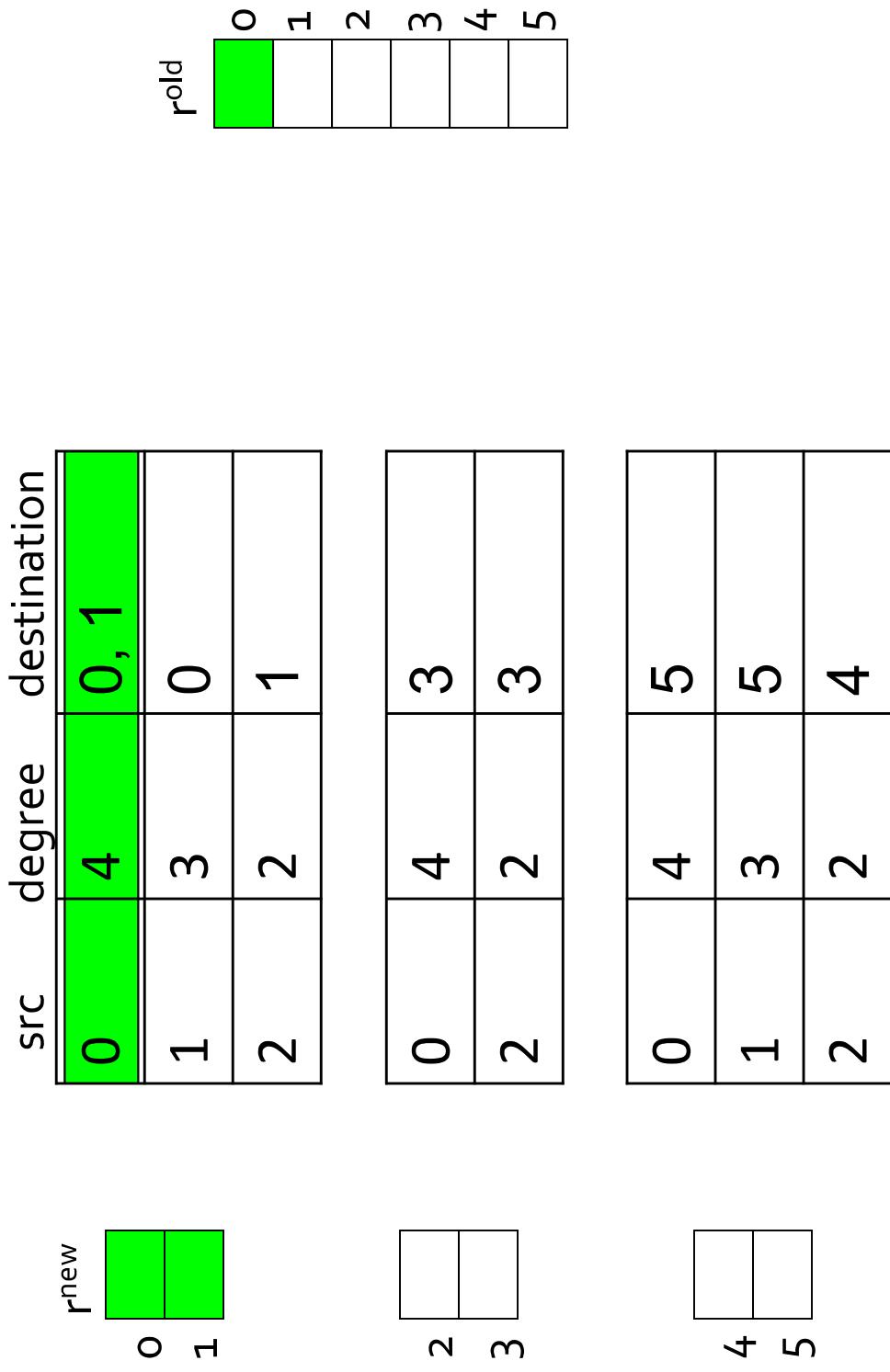


- Break  $r^{\text{new}}$  into  $k$  blocks that fit in memory
- Scan  $M$  and  $r^{\text{old}}$  Once for each block

# Analysis of Block Update

- **Similar to nested-loop join in databases**
  - Break  $r^{\text{new}}$  into  $k$  blocks that fit in memory
  - Scan  $M$  and  $r^{\text{old}}$  once for each block
- **Total cost:**
  - $k$  scans of  $M$  and  $r^{\text{old}}$
- **Cost per iteration of Power method:**
$$k(|M| + |r|) + |r| = k|M| + (k+1)|r|$$
- **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



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

# Block-Stripe Analysis

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